Package 'runjags'

November 20, 2010

Version 0.9.9-1

Date 2010-18-11

Title Run Bayesian MCMC Models in the BUGS syntax from Within R

Author Matthew Denwood <matthew.denwood@glasgow.ac.uk>

Maintainer Matthew Denwood <matthew.denwood@glasgow.ac.uk>

Depends R (>= 2.9), coda, lattice, stats, utils

Suggests rjags

SystemRequirements jags

Description A set of functions to allow any user specified model to be run in JAGS from within R, returning the MCMC chains as R objects. Includes functions to read external WinBUGS type textfiles, and allows several ways of automatically specifying model data from existing R objects or R functions. Also includes functions to automatically calculate model run length,autocorrealtion and Gelman Rubin statistic diagnostics for all models to simplify the process of achieving chain convergence. Designed for maximum compatibility with WinBUGS syntax,although minor modification to existing .bug files will be required. Also provides functions for submission and retrieval of jobs (both JAGS runs and user-specified R code execution) to Apple Xgrid distributed computing clusters (Mac OS X machines only). Requires Just Another Gibbs Sampler (JAGS) for most functions, see: http://www-fis.iarc.fr/~martyn/software/jags/

License GPL

URL http://cran.r-project.org/web/packages/runjags/

Repository CRAN

Date/Publication 2010-11-20 14:19:03

R topics documented:

ask . . . 2
autorun.jags . . . 3
autorun.jagsfile . . . 10
combine.mcmc . . . 12
dump.format . . . 14
findjags . . . 15
install.mgrid . . . 16
new_unique . . . 20
read.winbugs . . . 21
run.jags . . . 24
run.jagsfile . . . 30
runjags . . . 33
testjags . . . 34
timestring . . . 35
xgrid.run . . . 36
xgrid.run.jags . . . 45

Index **53**

`ask` *Obtain Input from User With Error Handling*

Description

A simple function to detect input from the user, and keep prompting until a response matching the class of input required is given.

Usage

```
ask(prompt="?", type="logical", bounds=c(-Inf, Inf),
   na.allow=FALSE)
```

Arguments

`prompt`	what text string should be used to prompt the user? (character string)
`type`	the class of object expected to be returned - "logical", "numeric", "integer", "character". If the user input does not match this return, the prompt is repeated
`bounds`	the lower and upper bounds of number to be returned. Ignored if type is "logical" or "character"
`na.allow`	if TRUE, allows the user to input "NA" for any type, which is returned as NA

Author(s)

Matthew Denwood <matthew.denwood@glasgow.ac.uk>

See Also

readline,

menu

Examples

```
#  Ask the user if they want to proceed
## Not run:
ask("Do you want to start the program now?", type="logical")

## End(Not run)
```

`autorun.jags` *Run a User Specified Bayesian MCMC Model in JAGS with Automatically*

Description

Runs a user specified JAGS (similar to WinBUGS) model from within R, returning a list of the MCMC chain(s) along with convergence diagnostics, autocorrelation diagnostics and monitored variable summaries. Chain convergence over the first run of the simulation is assessed using the Gelman and Rubin's convergence diagnostic. If necessary, the simulation is extended to improve chain convergence (up to a user-specified maximum time limit), before the required sample size of the Markov chain is calculated using the Raftery and Lewis's diagnostic. The simulation is extended to the required sample size dependant on autocorrelation and the number of chains.

This function is provided as an educational tool only, and is not a replacement for manually assessing convergence and Monte Carlo error for real-world applications. For more complex models, the use of run.jags directly with manual assessment of necessary run length is recommended. JAGS is called using the lower level function `run.jags`.

Usage

```
autorun.jags(model=stop("No model supplied"),
   monitor = stop("No monitored variables supplied"),
   data=NA, n.chains=2, inits = replicate(n.chains, NA),
   startburnin = 5000, startsample = 10000,
   psrf.target = 1.05, normalise.mcmc = TRUE,
   check.stochastic = TRUE, raftery.options = list(),
   crash.retry = 1, plots = TRUE, thin.sample = TRUE,
   jags = findjags(), silent.jags = FALSE,
   interactive=TRUE, max.time=Inf,
   adaptive=list(type="burnin", length=200), modules=c(""),
   factories=c(""), thin = 1, monitor.deviance = FALSE,
   monitor.pd = FALSE, monitor.pd.i = FALSE,
   monitor.popt = FALSE, keep.jags.files = FALSE, tempdir=TRUE,
   method=if(.Platform$OS.type=='unix' & .Platform$GUI!="AQUA" &
   Sys.info()['user']!='nobody') 'interruptible' else 'simple')
```

Arguments

`model`	a character string of the model in the JAGS language. No default.
`monitor`	a character vector of the names of variables to monitor. For all models, specifying 'deviance' as a monitored variable will calculate the model deviance, and 'dic' will calculate the Deviance Information Criterion (implies monitor.deviance etc, and requires more than 1 chain. No default.
`data`	either a named list or a character string in the R dump format containing the data. If left as NA, the model will be run without external data.
`n.chains`	the number of chains to use with the simulation. More chains will improve the sensitivity of the convergence diagnostic, but will cause the simulation to run more slowly. The minimum (and default) number of chains is 2.
`inits`	either a character vector with length equal to the number of chains the model will be run using, or a list of named lists representing names and corresponding values of inits for each chain. If a vector, each element of the vector must be a character string in the R dump format representing the initial values for that chain, or NA. If not all initialising variables are specified, the unspecified variables are sampled from the prior distribution by JAGS. Values left as NA result in all initial values for that chain being sampled from the prior distribution. The special variables '.RNG.seed', '.RNG.name', and '.RNG.state' are allowed for explicit control over random number generators in JAGS. Default NA.
`startburnin`	the number of initial updates to discard before sampling. Only used on the initial run before checking convergence. Default 5000 iterations.
`startsample`	the number of samples on which to assess convergence. More samples will give a better chance of allowing the chain to converge, but will take longer to achieve. Also controls the length of the pilot chain used to assess the required sampling length. The minimum is 4000 samples, which is the minimum required number of samples for a model with no autocorrelation and good convergence. Default 10000 iterations.
`psrf.target`	the value of the point estimate for the potential scale reduction factor of the Gelman Rubin statistic below which the chains are deemed to have converged (must be greater than 1). Default 1.05.
`normalise.mcmc`	the Gelman Rubin statistic is based on the assumption that the posterior distribution of monitored variables is roughly normal. For very skewed posterior distributions, it may help to log/logit transform the posterior before calculating the Gelman Rubin statistic. If normalise.mcmc == TRUE, the normality of the untransformed and log/logit transformed posteriors are compared for each monitored variable and the least skewed is used to calculate the Gelman Rubin statistic (this may take some time for large numbers of monitored variables). If FALSE, the data are left untransformed (this may give problems calculating the statistic in extreme cases). Default TRUE.
`check.stochastic`	non-stochastic monitored variables will cause errors when calculating the Gelman-Rubin statistic, if check.stochastic==TRUE then all monitored variables will be checked to ensure they are stochastic beforehand. This has a computational cost, and can be bypassed if check.stochastic==FALSE. Default TRUE.

`raftery.options` a named list which is passed as additional arguments to `raftery.diag`. Default none (default arguments to raftery.diag are used).

`crash.retry` the number of times to re-attempt a simulation if the model returns an error. Default 1 retry (simulation will be aborted after the second crash).

`plots` should traceplots and density plots be produced for each monitored variable? If TRUE, the returned list will include elements 'trace' and 'density' which consist of a list of lattice objects. The alternative is to use plot(results\$mcmc) to look at the density and traceplots for each variable using the traditional graphics system. Default TRUE.

`thin.sample` option to thin the final MCMC chain(s) before calculating summary statistics and returning the chains. Thinning very long chains allows summary statistics to be calculated more quickly. If TRUE, the chain is thinned to as close to a minimum of startsample iterations as possible (i.e. using a thinning interval of floor(chain.length/thin.sample) since the value must be an integer) and any excess iterations discarded to ensure the chain length matches thin.sample. If FALSE the chains are not thinned. A positive integer can also be specified as the desired chain length after thinning; the chains will be thinned to as close to this minimum value as possible. Default TRUE (thinned chains of length startsample returned). This option does NOT carry out thinning in JAGS, therefore R must have enough available memory to hold the chains BEFORE thinning. To avoid this problem use the 'thin' option instead.

`jags` the system call or path for activating JAGS. Default calls findjags() to attempt to locate JAGS on your system.

`silent.jags` should the JAGS output be suppressed? (logical) If TRUE, no indication of the progress of individual models is supplied. Default FALSE.

`interactive` option to allow the simulation to be interactive, in which case the user is asked if the simulation should be extended when run length and convergence calculations are performed and the extended simulation will take more than 1 minute. The function will wait for a response before extending the simulations. If FALSE, the simulation will be run until the chains have converged or until the next extension would extend the simulation beyond 'max.time'. Default FALSE.

`max.time` the maximum time for which the function is allowed to extend the chains to improve convergence, as a character string including units or as an integer in which case units are taken as seconds. Ignored if interactive==TRUE. If the function thinks that the next simulation extension to improve convergence will result in a total time of greater than max.time, the extension is aborted. The time per iteration is estimated from the first simulation. Acceptable units include 'seconds', 'minutes', 'hours', 'days', 'weeks', or the first letter(s) of each. Default "1hr".

`adaptive` a list of advanced options controlling the length of the adaptive mode of each simulation. Extended simulations do not require an adaptive phase, but JAGS prints a warning if one is not performed. Reduce the length of the adpative phase for very time consuming models. 'type' must be one of 'adaptive' or 'burnin'.

`modules` external modules to be loaded into JAGS. More than 1 module can be used. Default none.

`factories` factory modules to be loaded into JAGS. More than 1 factory can be used. Entried should be in the format '"<facname>" <status>, type(<factype>)', for example: factories='"mix::TemperedMix" off, type(sampler)'. Default none.

`thin` the thinning interval to be used in JAGS. Increasing the thinning interval may reduce autocorrelation, and therefore reduce the number of samples required, but will increase the time required to run the simulation. Using this option thinning is performed directly in JAGS, rather than on an existing MCMC object as with thin.sample. Default 1.

`monitor.deviance`
option to monitor the total deviance of the model using the DIC module in JAGS. If TRUE, an additional monitor called 'deviance' is added to the MCMC objects returned, representing the deviance of the model for each iteration and each chain. This option requires JAGS version 2 or greater. For more information see the JAGS user manual. Default FALSE.

`monitor.pd` option to monitor the total effective number of parameters in the model using the DIC module for JAGS. If TRUE, a 'pd' element is returned representing the total effective number of parameters at each iteration. This option requires JAGS version 2 or greater and at least 2 chains. For more information see the JAGS user manual. Default FALSE.

`monitor.pd.i` option to monitor the contribution of each parameter towards the total effective number of parameters using the DIC module for JAGS. If TRUE, a 'pd.i' element is returned representing the mean value for each parameter. This option requires JAGS version 2 or greater and at least 2 chains. For more information see the JAGS user manual. Default FALSE.

`monitor.popt` option to monitor the optimism of the expected deviance using the DIC module for JAGS. If TRUE, a 'popt' element is returned representing the mean value for each parameter. This option requires JAGS version 2 or greater and at least 2 chains. For more information see the JAGS user manual. Default FALSE.

`keep.jags.files`
option to keep the folder with files needed to call JAGS, rather than deleting it. May be useful for attempting to bug fix models. Since autorun.jags typically makes several calls to JAGS, all folders are kept - the order in which the folders were used can be ascertained from the creation dates of the files. Default FALSE.

`tempdir` option to use the temporary directory as specified by the system rather than creating files in the working directory. If keep.jags.files==TRUE then the folder is copied to the working directory after the job has finished (with a unique folder name based on 'runjagsfiles'). Any files created in the temporary directory are removed when the function exits for any reason. Default TRUE.

`method` the method with which to call JAGS; one of 'simple', 'interruptible' or 'parallel'. The former runs JAGS as a foreground process (the default behaviour for runjags < 0.9.6), 'interruptible' allows the JAGS process to be terminated immediately using the interrupt signal 'control-c' (terminal/console versions of R only), and 'parallel' runs each chain as a separate process on a separate core. Note that the latter uses separate JAGS instances to speed up execution of models with multiple chains (at the expense of using more RAM), but cannot be used with

monitor.pd, monitor.pd.i or monitor.popt. Each chain is specified using a different random number generator (.RNG.name) for up to 4 chains (the number of different RNG available in JAGS), unless .RNG.name is specified in the initial values. Because each chain uses a separate JAGS instance, JAGS has no way of ensuring independence between multiple chains using the same random number generator (as would normally be done when calling a single JAGS instance with multiple chains). Using more than 4 chains with the 'parallel' method without the use of new RNG factories may therefore produce dependence between chains, and is not recommended (a warning is given if trying to do so). Only the 'simple' method is available for Windows. On machines running Mac OS X and with access to an Apple Xgrid cluster, the method may be a list with an element 'xgrid.method="simple"' (see `xgrid.run.jags` for more information). Default 'interruptible' on terminal/console versions of R, or 'simple' on GUI versions of R or when running over xgrid (methods other than 'simple' require the use of 'ps' which is not available when running jobs as 'nobody' via xgrid).

Details

This function runs the specified model until the point estimate of the potential scale reduction factor of the Gelman-Rubin statistic for each monitored parameter is less than psrf.target (default 1.05). This is intended to make sure that the chains have converged before sampling from them (although this does not guarantee convergence so manual checking of traceplots is recommended). If convergence is not achieved within the time limit, then the function returns pilot.mcmc rather than mcmc, which will always be of length startsample since it is the unconverged mcmc object returned from the last run of the model. This chain is not suitable for making inference from, so a different name is used to avoid confusion. If convergence is achieved, the Raftery and Lewis's diagnostic is used to calculate the required number of samples based on the most heavily autocorrelated monitored variable. The chains are then extended to increase the sample size of each variable as necessary (so that sufficient samples are obtained from the COMBINED chains to satisfy the Raftery and Lewis's diagnostic) before being summarised and returned. Heavily autocorrelated models with large numbers of monitored variables may result in a required sample size larger than the available memory in R. If this is the case, try using the thin option to reduce autocorrelation (and therefore the required sample size) or monitor less variables.

Value

A list including the following elements:

`mcmc`	an MCMC list of MCMC objects representing the chains. Each MCMC object represents the value of each monitored variable (including the deviance, if specified) for that chain at each iteration. Renamed pilot.mcmc if the simulation is aborted before convergence or before the required sampling length is achieved
`end.state`	the end state of the last simulation extension performed. Can be used as initial values to extend the simulation further if required
`req.samples`	the minimum sample size required for the Markov chain as calculated using the Raftery and Lewis's diagnostic. This sample size is dependent on the thinning interval in JAGS

`samples.to.conv` the number of sampled iterations discarded due to poor convergence. The total number of iterations performed for the simulation is equal to req.samples + req.burnin + samples.to.conv (unless the simulation was aborted before reaching the required sampling length)

`thin` the thinning interval in JAGS used for the chains.

`summary` the summary statistics for the monitored variables from the combined chains. Renamed pilot.summary if pilot.only==TRUE or if the simulation is aborted before the required sampling length is achieved

`HPD` the 95% highest posterior density and median value of each monitored variable from the combined chains.

`psrf` the Gelman Rubin statistic for the monitored variables (similar to output of gelman.diag())

`autocorr` the autocorrelation diagnostic for the monitored variables (output of autocorr.diag())

`trace` a list of lattice objects representing traceplots for each monitored variable, of class 'plotindpages'. Calling each individual element will result in the traceplot for that variable being shown, calling the entire list will result in all traceplots being shown in new windows (which may cause problems with your R session if there are several monitored variables). To override the individual window plotting behaviour (to combine plots and/or save the plots to a file), either change the class of each object using for(i in 1:nvar(results$mcmc)) class(results$trace[[i]]) <- 'trellis' and then combine using the c.trellis method in the latticeExtra package, or simply re-generate the plots using the raw mcmc output. Not produced if plots==FALSE.

`density` a list of lattice objects representing density plots for each monitored variable, of class 'plotindpages'. Calling each individual element will result in the density plot for that variable being shown, calling the entire list will result in all density plots being shown in new windows (which may cause problems with your R session if there are several monitored variables). To override the individual window plotting behaviour (to combine plots and/or save the plots to a file), either change the class of each object using for(i in 1:nvar(results$mcmc)) class(results$density[[i]]) <- 'trellis' and then combine using the c.trellis method in the latticeExtra package, or simply re-generate the plots using the raw mcmc output. Not produced if plots==FALSE.

`pd` the total effective number of parameters in the model calculated using the DIC module for JAGS. Returned only if monitor.pd=TRUE is supplied to the function (or if 'dic' is specified as a monitored variable).

`pd.i` the contribution of each parameter towards the total effective number of parameters calculated using the DIC module for JAGS. Returned only if monitor.pd.i=TRUE is supplied to the function.

`popt` the optimism of the expected deviance calculated using the DIC module for JAGS. Returned only if monitor.popt=TRUE is supplied to the function (or if 'dic' is specified as a monitored variable).

`dic` The Deviance Information Criterion (DIC) model fit statistics. Returned only if 'dic' is specified as a monitored variable. The default print method displays the

DIC calculated using both pd (dic) and popt (ped), using the deviance from the combined chains and for each individual chain. Separate values for the mean pd, sum popt and mean deviance is also listed. For more information on the types of DIC calculated, see the JAGS manual.

Author(s)

Matthew Denwood <matthew.denwood@glasgow.ac.uk>

See Also

run.jags,

autorun.jagsfile,

combine.mcmc,

raftery.diag,

gelman.diag,

autocorr.diag

Examples

```
# run a model to calculate the intercept and slope of the expression
# y = m x + c, assuming normal observation errors for y:

## Not run:
# Simulate the data
x <- 1:100
y <- rnorm(length(x), 2*x + 10, 1)

# Model in the JAGS format
model <- "model {
for(i in 1 : N){
Y[i] ~ dnorm(true.y[i], precision);
true.y[i] <- (m * X[i]) + c;
}
m ~ dunif(-1000,1000);
c ~ dunif(-1000,1000);
precision ~ dexp(1);
}"

# Convert the data to a named list
data <- list(X=x, Y=y, N=length(x))

# Run the model
results <- autorun.jags(model=model, monitor=c("m", "c", "precision"),
data=data)

# Analyse traceplots of the results to assess convergence:
results$trace

# Summary of monitored variables:
```

```
results$summary

## End(Not run)
```

`autorun.jagsfile` *Read a User Specified Model in a WinBUGS Type Textfile or Character*

Description

Runs a user specified JAGS (similar to WinBUGS) model in a WinBUGS type Textfile from within R, returning a list of the MCMC chain(s) along with convergence diagnostics, autocorrelation diagnostics and monitored variable summaries. This function is a wrapper for `run.jagsfile` with autorun==TRUE, and uses `read.winbugs` to extract model and data definitions from the specified file or string.

Usage

```
autorun.jagsfile(path=stop("No path or model string supplied"),
   datalist=NA, initlist=NA, n.chains=NA, data=NA, model=NA,
   inits=NA, monitor=NA, call.jags=TRUE, ...)
```

Arguments

`path`	either a relative or absolute path to a textfile (including the file extension) containing a model in the JAGS language and possibly monitored variable names, data and/or initial values, or a character string of the same. May also be a vector of paths to different text files, possibly separately containing the model, data and intitial values. No default. The model must be started with the string 'model{' and ended with '}' on new lines. Data must be similarly started with 'data{', monitored variables with 'monitor{', and initial values as 'inits{', and all ended with '}'. If multiple models are found, all but the first one are ignored with a warning. Multiple data blocks and monitor blocks are combined, multiple inits blocks are used for different chains. The model block may also contain automatically generated data and initial values variables using '#data# variable' and '#inits# variable', and more monitored variables using '#monitor# variable'. See `read.winbugs` for more information.
`datalist`	an optional named list containing variables used as data, or alternatively a function (with no arguments) that returns a named list. If any variables are specified in the model block using '#data# variable', the value for the corresponding named variable is taken from datalist if present (or the result of datalist() if specified as a function which is useful for specifying randomly generated data), or the parent environment, or finally the global environment if not found anywhere else. Ignored if '#data# variable' is not used in the model block. Default NA.
`initlist`	an optional named list containing variables used as initial values, or alternatively a function (with a single argument representing the chain number) that returns a named list. If any variables are specified in the model block using '#inits# variable', the value for the corresponding named variable is taken from initlist if

present (or the result of datalist(chain.no) if specified as a function which allows both randomly generated initial values and different values for each chain), or the parent environment, or finally the global environment if not found anywhere else. Ignored if '#inits# variable' is not used in the model block. Note: different chains are all given the same starting values if specified as a named list or taken from any envirnoment; if different values are desired for each chain initlist should be specified as a function. Default NA.

`n.chains` the number of chains to use with the simulation. More chains will improve the sensitivity of the convergence diagnostic, but will cause the simulation to run more slowly. If NA, the number of chains will be taken from the number of inits blocks in the model file. If NA and no inits blocks are found, 2 chains are used with a warning. Default NA.

`data` OPTIONAL character vector in the R dump format (or named list) containing the data. If supplied (!=NA), all data in the model file is ignored. Default NA.

`model` OPTIONAL model in JAGS syntax. If supplied (!=NA), the model in the model file is ignored. Default NA.

`inits` OPTIONAL character vector(s) in the R dump format containing the initial value(s). If supplied (!=NA), all inits in the model file are ignored. Default NA.

`monitor` OPTIONAL character vector containing the monitored variables. If supplied (!=NA), all monitor statements in the model block are ignored. Default NA.

`call.jags` option results in either simulation being called if TRUE, or returns a named list of the data, model, initial values, monitored variables and number of chains (which can be supplied to autorun.jags) if FALSE.

`...` other options to be passed directly to autorun.jags (see `autorun.jags`).

Value

The output of autorun.jags. See the help file `autorun.jags` for more information.

Author(s)

Matthew Denwood `<matthew.denwood@glasgow.ac.uk>`

See Also

`run.jagsfile`,

`autorun.jags`,

`run.jags`,

`read.winbugs`

Examples

```
# run a model to calculate the intercept and slope of the expression
#  y = m x + c, assuming normal observation errors for y:

## Not run:
```

```
# Model in the JAGS format
model <- "model {
for(i in 1 : N){ #data# N
Y[i] ~ dnorm(true.y[i], precision); #data# Y
true.y[i] <- (m * X[i]) + c; #data# X
}
m ~ dunif(-1000,1000); #inits# m
c ~ dunif(-1000,1000);
precision ~ dexp(1);
#monitor# m, c, precision, dic
}"

# Simulate the data
X <- 1:100
Y <- rnorm(length(X), 2*X + 10, 1)
N <- length(X)

initfunction <- function(chain) return(switch(chain, "1"=list(m=-10),
"2"=list(m=10)))

results <- autorun.jagsfile(model, n.chains=2, initlist=initfunction)

# Analyse traceplots of the results to assess convergence:
results$trace

# Analyse the results
results$summary

## End(Not run)
```

`combine.mcmc` *Combine Two or More MCMC Objects With the Same Number of Chains Into*

Description

Allows an MCMC object (with 1 or more chains) to be combined with another (or several other) MCMC object(s) representing extensions of the same simulation, to produce one MCMC object that contains the continuous combined Markov chains of the other MCMC objects. Also provides a safe way to thin a single MCMC object or list.

Usage

```
combine.mcmc(mcmc.objects=list(), thin=1, return.samples=NA,
collapse.chains=if(length(mcmc.objects)==1) TRUE else FALSE)
```

Arguments

mcmc.objects	a list of MCMC objects, all with the same number of chains, or a single MCMC object or list. No default.
thin	an integer to use to thin the (final) MCMC object by, in addition to any thinning already applied to the objects before being passed to combine.mcmc. Ignored if return.samples is specified (!is.na). Default 1 (no additional thinning is performed).
return.samples	the number of samples to return after thinning. The chains will be thinned to as close to this minimum value as possible, and any excess iterations discarded. Supersedes thin if both are specified. Ignored if niter(mcmc.objects) < return.samples. Default NA.
collapse.chains	option to combine all MCMC chains into a single MCMC chain with more iterations. Can be used for combining chains prior to calculating results in order to reduce the Monte Carlo error of estimates. Default TRUE if a single mcmc.object is provided, or FALSE otherwise.

Value

A single MCMC object of length equal to the sum of the lengths of the input MCMC objects (if thin=1)

Author(s)

Matthew Denwood <matthew.denwood@glasgow.ac.uk>

See Also

run.jags,

testjags

Examples

```
# run a model, then extend the simulation and combine the two MCMC
#  objects, with thinning to 5000 samples.

## Not run:

# Model in the JAGS format
model <- "model {
for(i in 1 : N){ #data# N
Y[i] ~ dnorm(true.y[i], precision); #data# Y
true.y[i] <- (m * X[i]) + c; #data# X
}
m ~ dunif(-1000,1000);
c ~ dunif(-1000,1000);
precision ~ dexp(1);
```

```
#monitor# m, c, precision
}"

# Simulate the data
X <- 1:100
Y <- rnorm(length(X), 2*X + 10, 1)
N <- length(X)

results1 <- run.jagsfile(model, n.chains=2, burnin=5000, sample=10000)
results2 <- run.jagsfile(model, inits=results1$end.state, burnin=0,
sample=10000)

results <- combine.mcmc(list(results1$mcmc, results2$mcmc),
return.samples=5000)

# Analyse the results
summary(results)

## End(Not run)
```

`dump.format` *Produce a Character String in the R Dump Format to Be Used With Jags*

Description

Convert a named list of numeric vector(s) or array(s) of data or initial values to a character string in the correct format to be read by JAGS as either data or initial values, using the run.jags function.

Usage

```
dump.format(data=list())
```

Arguments

`data`	A named list of numeric or integer (or something that can be coerced to numeric) vectors, matrices or arrays. The name of each list item will be used as the name of the resulting dump.format variables. Alternatively, if the list is of length 2 and is not named, and the first element is a character string, the first element will be used as the name for the second element.

Details

This function creates a character string of the supplied variables in the same way that dump() would, except that the result is returned as a character string rather than written to file. Additionally, dump.format() will look for any variable with the name '.RNG.name' and double quote the value if not already double quoted (to ensure compatibility with JAGS).

Value

A character string in the R dump format.

Author(s)

Matthew Denwood <matthew.denwood@glasgow.ac.uk>

See Also

run.jags

Examples

```
initial.values.1 <- dump.format(list(mean="1", sd="0.1",
lambda=matrix(1, ncol=3, nrow=10)))
initial.values.2 <- dump.format(list(mean="10", sd="10",
lambda=matrix(10, ncol=3, nrow=10)))
data <- dump.format(list(N="10", Count=c(4,2,7,0,6,9,1,4,12,1)))
```

---

findjags *Attempt to Locate a JAGS Install*

Description

Search the most likely locations for JAGS to be installed on the users system, based on the operating system, and return the most likely path to try. Where multiple installs exist, findjags will attempt to return the path to the install with the highest version number. For Unix systems, calling jags using 'jags' requires the jags binary to be in the search path, which may be specified in your user '.Profile' if necessary (the JAGS executable is also looked for in the default install location of /usr/local/bin/jags if popen support is enabled).

Usage

```
findjags(ostype = .Platform$OS.type)
```

Arguments

ostype	the operating system type. There is probably no reason to want to change this...

Value

A path or command for the most likely location of JAGS on the system. On unix this will always be 'jags', on Windows for example "C:/Program Files/JAGS/bin/jags-terminal.exe" or "C:/Program Files/JAGS/JAGS-1.0.0/bin/jags-terminal.exe"

Author(s)

Matthew Denwood <matthew.denwood@glasgow.ac.uk>

See Also

testjags,

run.jags

Examples

```
findjags()
```

---

install.mgrid *Install the mgrid script to extend the functionality of the Apple Xgrid*

Description

This function installs the mgrid script included in the runjags package to /usr/local/bin/mgrid to allow the multiple task features of the xgrid related functions to be used. The mgrid script also provides other useful options, see the details section. This function will only run on Mac OS X, and is only useful for those wishing to access an Xgrid distributed computing cluster.

Please note that installation of the mgrid script will require an administrators password. If you are uncomfortable with entering this into an R function, then you can manually install the mgrid script by opening a Terminal at the runjags/xgrid folder and typing 'sudo chmod 755 mgrid.sh' followed by 'sudo cp mgrid.sh /usr/local/bin/mgrid'

Usage

```
install.mgrid(libpath=.Library, ask=TRUE)
```

Arguments

libpath	the path to the library where runjags is installed. Unless specified when installing the package, this will be the default provided by .Library.
ask	option to confirm the old and new versions of mgrid and ask to continue before overwriting the old version with the new version. Default TRUE.

Details

The xgrid related functions within runjags allow arbitrary R code to be run on Xgrid distributed computing clusters from within R. All the functionality could be replicated by saving all necessary objects to files and using the Xgrid command line utility to submit and retrieve the job manually; these functions merely provide the convenience of not having to do this manually. Xgrid support is only available on Mac OS X machines.

All functions can be run using the built-in xgrid commands, however some added functionality (including multi-tasks jobs to enable the 'separatetasks' method) is provided by the 'mgrid.sh' BASH shell script which is included with the runjags package (in the 'inst/xgrid' folder for the package source or the 'xgrid' folder for the installed package). This function automatically installs the mgrid script from this location.

The xgrid controller hostname and password must be set as environmental variables. The command line version of R knows about environmental variables set in the .profile file, but unfortunately the GUI version does not and requires them to be set from within R using:

Sys.setenv(XGRID_CONTROLLER_HOSTNAME="<hostname>")

Sys.setenv(XGRID_CONTROLLER_PASSWORD="<password>")

(These lines could be copied into your .Rprofile file for a 'set and forget' solution)

You can look at the possible arguments to mgrid by typing 'mgrid' in the Terminal or system('mgrid') within R. The output is also shown below for convinience:

mgrid – version 3.01, June 2010

by Matthew Denwood

SYNOPSIS

A bash script that provides a replacement for xgrid -job submit using batch file submission with support for multi-task jobs, scheduler hinting, environmental variables, and built-in ART scripts for selection of intel or ppc machines.

USAGE

mgrid [-s stdin] [-i indir] [-d jobid] [-e email-address] [-a art-path | -z art-path] [-b batchname] [-c arch] [-v "env_variable=env_value [...]"] [-q] [-h node_name] [-f] [-r ram_required_(MB)] [-n name] [-t number_of_tasks] cmd [arg1 [...] ['$task']]

mgrid -l [-m] [-u]

mgrid -?

OPTIONS

The following options are equivaent to those given for xgrid -job submit:

-s Use supplied file as standard input (equivalent to -si)

-i Input supplied directory (equivalent to -in)

-d Wait for the dependant job ID to finish (equivalent to -dids, with the limitation that only 1 dependant job can be specified)

-e Use supplied email address to report status changes (equivalent to -email)

-a Use supplied file as an ART script (equivalent to -art)

Other options for xgrid -job submit are not supported. The following options are unique for this script:

-z Use supplied file as an ART script, and also calculate the node ranking scores using the supplied script rather than the inbuilt script or a script in 'Application Support/mgrid/scoring.rb'. The disadvantage of using an ART script to calculate node ranking in this way is that for multi-task jobs, the score is not altered to take into account previous task allocations. For an alternative method which preserves this feature see: 'runjags/inst/xgrid/node_scoring_example.rb'. For users outside the University of Glasgow or not using node ranking, the -r option is identical to the -a option. Supplying

" " or "none" as the argument. disables node ranking and the ranking ART script for submission of this job.

-b Produce a batch file with the specified file name and stop (does not submit to xgrid). This option also prints the (ranked) scores and nodes to screen. If the batchname specified is "profile" (or "/dev/null") then more detailed information about the scoring is shown, no batch file is produced and cmd is ignored (and may be omitted).

-c Ensure that jobs are run only on intel or ppc machines - should be 'intel' or 'ppc'.

-v A space seperated string of environmental variable(s) in the form of "var_one=value_one var_two=value_two" to be set locally before running the command.

-q Wait for nodes running OS X server to become available rather than running jobs on desktop nodes.

-h Don't run the ranking script, and use the provided node name(s) instead. For multiple tasks, separate node names with a colon (no space). Entire string must be quoted if any node name contains a space. Colons in node names will produce either an error or unexpected results. If the node name does not match any of the available nodes then the first available node will be chosen (unless the -f option is also specified in which case the job/task will hang). Supplying a blank node name (-h) produces a job or batch file with no schedule hinting (this also effectively turns off node ranking for University of Glasgow users, although the ranking script is still passed as an ART script to the job - see the -r option for a way to disable both).

-f Force the controller to assign the job to the highest ranked (or specified using -t) node by using an ART script rather than schedule hinting. Note that ALL tasks will be run on the same node.

-r The minimum amount of free RAM in MB that is required for the job. If not supplied, 10MB is used as a default amount to prevent machines that have free processors but no free RAM from accepting jobs.

-n The name to give the job (appears on xgrid admin etc). If none is supplied, the command is used as the name.

-t The number of tasks being run. The arguments should include one containing '$task' which denotes the task number (this MUST be enclosed in single quotes or the $ sign escaped), otherwise the task number will be passed as the last argument to the command.

-l Display a list of jobs currently on xgrid and exit. The following two arguments can also be given:

-m Include the current status of the jobs.

-u Include the username and hostame of whoever submitted the job (provided the jobs were submitted using mgrid).

All other arguments are ignored.

-? Print a help/usage message and exit. All other arguments are ignored.

ARGUMENTS

cmd is the command to be run on xgrid, and the remaining arguments are passed as arguments to this command. The special argument '$task' is used to denote the task number, and is appended to (any) other commands if ntasks is specified (even if it is 1) and '$task' is not found among the other arguments. The '$task' variable can be embedded in other text to form an argument that changes with the task number, for example mgrid -t 2 /usr/bin/cal -y '200$task' would print 2001 for task 1 and 2002 for task 2. *NB* If using the '$task' special variable in this way, ensure that the argument is enclosed in single quotes (NOT double quotes), or use backslash to escape it (as in "\$"), as the shell will otherwise evaluate "$task" to "" on passing the argument to mgrid.

REQUIREMENTS

Requires XGRID_CONTROLLER_HOSTNAME and XGRID_CONTROLLER_PASSWORD to be set as environmental variables (these cannot currently be specified as arguments).

ABOUT

This script is a replacement for xgrid -job submit that provides some extra features, including support for jobs with multiple tasks, use of environmental variables and easier targeting of the job to named nodes or ppc vs intel machines. For use within the University of Glasgow only, the script also uses schedule hints to automatically target nodes with higher ART scores. This is basically an ugly workaround for the fact that ART doesn't (currently) rank nodes correctly when prioritising jobs. It is possible to implement this feature for use with other xgrid setups; please email me if you are interested in doing so. Please also feel free to email me if you would be willing to clean up and add better documention to my woefully cobbled together code so that it might be more useful to others....

For use with hydra.vet.gla.ac.uk only:

By default, an inbuilt script is used to rank the available nodes for job/task allocation. This scoring is performed based on the number of processors per node, the number of jobs already running, and whether or not the node is a server (running Mac OS X server). The first task is assigned to the node with the highest score before incrementing the number of jobs running on that node to account for the extra task, and re-calculating the score for that node. This process is then repeated for all remaining tasks. Additional information detailing the amount of RAM, processor speed and 32/64 bit availability on the nodes is collected but not used by default, except for the amount of RAM free which is used to score a node 0 if it is less than the given minimum. To change the way that tasks are distributed among the available nodes, modify the code as desired in the file included with the runjags package (found in: 'runjags/inst/xgrid/node_scoring_example.rb'), rename the file either 'scoring.rb' or 'scoring.sh', and save either in the working directory or in '/Library/Application Support/mgrid/'. Alternatively, a simple scoring script can be specified using the -r option, however this score will not be updated after assigning each task to account for the extra task, so all tasks will be assigned to the node with the highest score at the start. If the number of tasks exceeds the number of available processors on that node, the remaining tasks will be allocated to nodes chosen by the xgrid controller. For jobs with only a single task this option should produce the same results as the more complex method described above.

NOTES

This script is distributed 'as is', both FREELY and WITHOUT CHARGE, under the GNU general public license (see http://www.gnu.org/copyleft/gpl.html). I am therefore not liable for any damage to your computer, xgrid cluster, or sanity caused by using it.

Value

The version of mgrid installed is returned invisibly.

Author(s)

Matthew Denwood <matthew.denwood@glasgow.ac.uk>

See Also

xgrid.run.jags for functions to run JAGS models on Xgrid, and xgrid.run for running arbitrary functions on Xgrid.

`new_unique` *Create a Unique Filename*

Description

Search the current working directory for a file or directory matching the input name, and if it exists suggest a new name by appending a counter to the input name. Alternatively, the function can ask the user if the existing file should be overwritten, in which case the existing file will be erased if the answer is 'yes'. The function also checks for write access permissions at the current working directory.

Usage

```
new_unique(name = NA, suffix = "", ask = FALSE,
prompt = "A file or directory with this name already exists.  Overwrite?",
touch=FALSE, type='file')
```

Arguments

`name`	the filename to be used (character string). A vector of character strings is also permissible, in which case they will be pasted together. One or more missing (NA) values can also be used, which will be replaced with a randomly generated 9 character alphanumeric string. Default NA.
`suffix`	the file extension (including '.') to use (character string). If this does not start with a '.', one will be prepended automatically. Default none.
`ask`	if a file exists with the input name, should the function ask to overwrite the file? (logical) If FALSE, a new filename is used instead and no files will be over-written. Default FALSE.
`prompt`	what text string should be used to prompt the user? (character string) Ignored is ask==FALSE. A generic default is supplied.
`touch`	option to create (touch) the file/folder after generating the unique name, which prevents other processes from sneaking in and creating a file with the same name before the returned filename has had chance to be used. Default FALSE.
`type`	if touch==TRUE, then type controls if a file or directory is created. One of 'file', 'f', 'directory', or 'd'. Defult 'file'.

Value

A unique filename that is safe to use without fear of destroying existing files

Author(s)

Matthew Denwood <matthew.denwood@glasgow.ac.uk>

See Also

ask

Examples

```
#  Create a file name that is unlikely to exist already,
#  with a .R extension.
new_unique(c("new_file", NA), ".R", ask=FALSE)
```

---

read.winbugs *Extract Any Models, Data, Monitored Variables or Initial Values As*

Description

Read a user specified WinBUGS type textfile or character variable and extract any models, data, monitored variables or initial values as character vectors. Used by (auto)run.jagsfile to interpret the input file(s) or strings.

Usage

```
read.winbugs(path)
```

Arguments

path	either a relative or absolute path to a textfile (including the file extension) containing a model in the JAGS language and possibly monitored variable names, data and/or initial values, or a character string of the same. May also be a vector of paths to different text files, possibly separately containing the model, data and intitial values. No default. The model must be started with the string 'model{' and ended with '}' on new lines. Data must be similarly started with 'data{', monitored variables with 'monitor{', and initial values as 'inits{', and all ended with '}'. Seperate variables in such blocks must be separated by a line break. If multiple models are found, all but the first one are ignored with a warning. Multiple data blocks and monitor blocks are combined, multiple inits blocks are used for different chains. Monitors may also be given using the phrase '#monitor# variable' within the model block, in which case 'variable' is added to the list of monitored variables found in the monitor block(s). The use of automatically generated data and initial values is also supported using similar syntax, with '#data# variable' for automatically generated data variables or '#inits# variable' for automatically generated initial value variables in which case 'variable' is used as data or initial values with a value taken by run.jagsfile from

datalist, initlist or R objects as appropriate. '#inits#', '#data#' and '#monitor#' statements can appear on the same line as model code, but no more than one of these statements should be used on the same line. Examples of acceptable model syntax is given below.

Value

A named list of 'model' containing the model description, 'data' containing the data given in the data block(s), 'autodata' containing data variables specified using '#data#' in the model block, 'inits' containing the initial values given in the initial value block(s), 'autoinits' containing initial value variables specified using '#inits#' in the model block, and 'monitor' containing the monitored variables specified in the monitor blocks and by using '#monitor#' within the model block. This function is specified primarily for WinBugs compatibility, so data blocks would normally contain the data in a list format rather than the code format that is allowed in JAGS to perform data transformations (see JAGS manual section 7.0.5). These JAGS format data blocks can be specified, and the function will attempt to differentiate the two types of data from the presence of syntactical cues such as square brackets, for loops, 'list' and .Dim structural assignments. If none of these are found, the data block is assumed to be a WinBugs type data block and is passed to JAGS as data. This behaviour can be over-ridden by inserting '#jagsdata#' or '#bugsdata#' into the data block as appropriate. More than one data block is allowed, and each will be differentiated independently.

Author(s)

Matthew Denwood <matthew.denwood@glasgow.ac.uk>

See Also

run.jagsfile

Examples

```
## Not run:

# ALL SYNTAX GIVEN BELOW IS EQUIVALENT

# Use a modified WinBUGS text file with manual inits and manual data and
# a seperate monitor block (requires least modification from a WinBUGS
# file).  For compatibility with WinBUGS, the use of list() to enclose
# data and initial values is allowed and ignored, however all seperate
# variables in the data and inits blocks must be seperated with a line
# break (commas or semicolons before linebreaks are ignored).  'data{'
# and 'inits{' must also be added to WinBUGS textfiles so that the
# function can seperate data from initial values.  Iterative loops are
# allowed in data blocks but not in init blocks.  See also the differences
# in JAGS versus WinBUGS syntax in the JAGS help file.

# Contents of a textfile 'mymodel.bug':
```

```
model{

for(i in 1:N){
Count[i] ~ dpois(mean)
}
mean ~ dgamma(0.01, 100)
}

data{
list(Count <- c(1,2,3,4,5,6,7,8,9,10),
N <- 10)

}
inits{
list(
mean <- 1)
}

inits{
list(
mean <- 100)
}

monitor{
mean
}

# end text file

read.winbugs('pathtofile/mymodel.bug')


# Use internal character variable, define monitors in the model,
# use autodata and manual initial values:

string <- "
model{

for(i in 1:N){
Count[i] ~ dpois(mean) #data# Count, N
}
mean ~ dgamma(0.01, 100)
#monitor# mean
}

inits{
mean <- 1
}

inits{
mean <- 100
}
"
```

```
read.winbugs(string)


# Use autoinits and a mixture of manual and autodata:
string <- "
model{

for(i in 1:N){
Count[i] ~ dpois(mean) #data# Count
}
mean ~ dgamma(0.01, 100)
#monitor# mean
#inits# mean
}

data{

N <- 10

}
"

read.winbugs(string)


## End(Not run)
```

`run.jags` *Run a User Specified Bayesian MCMC Model in JAGS from Within R*

Description

Runs a user specified JAGS (similar to WinBUGS) model from within R, returning a list of the MCMC chain(s) along with convergence diagnostics, autocorrelation diagnostics and monitored variable summaries. Data and initial values can either be supplied in the R dump format (see dump.format() for an easy way to do this), or as a named list. A character vector of variables to monitor must also be supplied.

Requires Just Another Gibbs Sampler (JAGS), see http://www-fis.iarc.fr/~martyn/software/jags/. The GUI interface for R in Windows may not continually refresh the output window, making it difficult to track the progress of the simulation (if silent.jags is FALSE). To avoid this, you can run the function from the terminal version of R (located in the Program Files/R/bin/ folder).

Usage

```
run.jags(model=stop("No model supplied"),
monitor = stop("No monitored variables supplied"), data=NA, n.chains=2,
inits = replicate(n.chains, NA), burnin = 5000*thin,
sample = 10000*thin, adapt=if(burnin<200) 100 else 0, jags = findjags(),
silent.jags = FALSE, check.conv = TRUE, plots = TRUE,
psrf.target = 1.05, normalise.mcmc = TRUE, check.stochastic = TRUE,
modules=c(""), factories=c(""), thin = 1, monitor.deviance = FALSE,
monitor.pd = FALSE, monitor.pd.i = FALSE, monitor.popt = FALSE,
keep.jags.files = FALSE, tempdir=TRUE, method=if(.Platform$OS.type=='unix'
& .Platform$GUI!="AQUA" & Sys.info()['user']!='nobody') 'interruptible'
else 'simple')
```

Arguments

model	a character string of the model in the JAGS language. No default.
monitor	a character vector of the names of variables to monitor. For all models, specifying 'deviance' as a monitored variable implies monitor.deviance, and 'dic' will calculate the Deviance Information Criterion (implies monitor.deviance/monitor.pd/monitor.popt and requires more than 1 chain). No default.
data	a character string in the R dump format (or a named list) containing the data. If left as NA, no external data is used in the model. Default NA.
n.chains	the number of chains to use for the simulation. Must be a positive integer. Default 2.
inits	either a character vector with length equal to the number of chains the model will be run using, or a list of named lists representing names and corresponding values of inits for each chain. If a vector, each element of the vector must be a character string in the R dump format representing the initial values for that chain, or NA. If not all initialising variables are specified, the unspecified variables are sampled from the prior distribution by JAGS. Values left as NA result in all initial values for that chain being sampled from the prior distribution. The special variables '.RNG.seed', '.RNG.name', and '.RNG.state' are allowed for explicit control over random number generators in JAGS. Default NA.
burnin	the number of burnin iterations (not sampled) to use (numeric). Default 5000 iterations.
sample	the number of sampling iterations to use (numeric). Default 10000 iterations.
adapt	advanced option to control the length of the adaptive phase directly, which is otherwise half the length of the burnin period. Default is 0, unless burnin is less than 200 in which case 100 adapitve iterations are used.
jags	the system call or path for activating JAGS. Default calls findjags() to attempt to locate JAGS on your system.
silent.jags	should the JAGS output be suppressed? (logical) If TRUE, no indication of the progress of individual models is supplied. Default FALSE.

`check.conv` should the convergence be assessed after the model has completed? If TRUE, each monitored variable will be assessed for a potential scale reduction factor of the Gelman Rubin statistic of less than 1.05, which indicates adequate convergence. 2 or more chains are required to assess convergence. Default TRUE.

`plots` should traceplots and density plots be produced for each monitored variable? If TRUE, the returned list will include elements 'trace' and 'density' which consist of a list of lattice objects. The alternative is to use plot(results$mcmc) to look at the density and traceplots for each variable using the traditional graphics system. Default TRUE.

`psrf.target` the value of the point estimate for the potential scale reduction factor of the Gelman Rubin statistic below which the chains are deemed to have converged (must be greater than 1). Ignored if check.conv==FALSE. Default 1.05.

`normalise.mcmc` the Gelman Rubin statistic is based on the assumption that the posterior distribution of monitored variables is roughly normal. For very skewed posterior distributions, it may help to log/logit transform the posterior before calculating the Gelman Rubin statistic. If normalise.mcmc == TRUE, the normality of the untransformed and log/logit transformed posteriors are compared for each monitored variable and the least skewed is used to calculate the Gelman Rubin statistic. If FALSE, the data are left untransformed (this may give problems calculating the statistic in extreme cases). Ignored if check.conv==FALSE. Default TRUE.

`check.stochastic` non-stochastic monitored variables will cause errors when calculating the Gelman-Rubin statistic, if check.stochastic==TRUE then all monitored variables will be checked to ensure they are stochastic beforehand. This has a computational cost, and can be bypassed if check.stochastic==FALSE. Default TRUE.

`modules` external modules to be loaded into JAGS. More than 1 module can be used. Default none.

`factories` factory modules to be loaded into JAGS. More than 1 factory can be used. Entried should be in the format '"<facname>" <status>, type(<factype>)', for example: factories='"mix::TemperedMix" off, type(sampler)'. Default none.

`thin` the thinning interval to be used in JAGS. Increasing the thinning interval may reduce autocorrelation, and therefore reduce the number of samples required, but will increase the time required to run the simulation. Default 1.

`monitor.deviance` option to monitor the total deviance of the model using the DIC module in JAGS. If TRUE, an additional monitor called 'deviance' is added to the MCMC objects returned, representing the deviance of the model for each iteration and each chain. This option requires JAGS version 2 or greater. For more information see the JAGS user manual. Default FALSE.

`monitor.pd` option to monitor the total effective number of parameters in the model using the DIC module for JAGS. If TRUE, a 'pd' element is returned representing the total effective number of parameters at each iteration. This option requires JAGS version 2 or greater and at least 2 chains. For more information see the JAGS user manual. Default FALSE.

`monitor.pd.i` option to monitor the contribution of each parameter towards the total effective number of parameters using the DIC module for JAGS. If TRUE, a 'pd.i' element is returned representing the mean value for each parameter. This option requires JAGS version 2 or greater and at least 2 chains. For more information see the JAGS user manual. Default FALSE.

`monitor.popt` option to monitor the optimism of the expected deviance using the DIC module for JAGS. If TRUE, a 'popt' element is returned representing the mean value for each parameter. This option requires JAGS version 2 or greater and at least 2 chains. For more information see the JAGS user manual. Default FALSE.

`keep.jags.files` option to keep the folder with files needed to call JAGS, rather than deleting it. May be useful for attempting to bug fix models. Default FALSE.

`tempdir` option to use the temporary directory as specified by the system rather than creating files in the working directory. If keep.jags.files==TRUE then the folder is copied to the working directory after the job has finished (with a unique folder name based on 'runjagsfiles'). Any files created in the temporary directory are removed when the function exits for any reason. Default TRUE.

`method` the method with which to call JAGS; one of 'simple', 'interruptible' or 'parallel'. The former runs JAGS as a foreground process (the default behaviour for runjags < 0.9.6), 'interruptible' allows the JAGS process to be terminated immediately using the interrupt signal 'control-c' (terminal/console versions of R only), and 'parallel' runs each chain as a separate process on a separate core. Note that the latter uses separate JAGS instances to speed up execution of models with multiple chains (at the expense of using more RAM), but cannot be used with monitor.pd, monitor.pd.i or monitor.popt. Each chain is specified using a different random number generator (.RNG.name) for up to 4 chains (the number of different RNG available in JAGS), unless .RNG.name is specified in the initial values. Because each chain uses a separate JAGS instance, JAGS has no way of ensuring independence between multiple chains using the same random number generator (as would normally be done when calling a single JAGS instance with multiple chains). Using more than 4 chains with the 'parallel' method without the use of new RNG factories may therefore produce dependence between chains, and is not recommended (a warning is given if trying to do so). Only the 'simple' method is available for Windows. On machines running Mac OS X and with access to an Apple Xgrid cluster, the method may be a list with an element 'xgrid.method="simple"' (see `xgrid.run.jags` for more information). Default 'interruptible' on terminal/console versions of R, or 'simple' on GUI versions of R or when running over xgrid (methods other than 'simple' require the use of 'ps' which is not available when running jobs as 'nobody' via xgrid).

Value

The results of the simulation are returned as list including the following items (omitting some items if check.conv==FALSE):

`mcmc` an MCMC list of MCMC objects representing the chains. Each MCMC object represents the value of each monitored variable (including the deviance, if specified) for that chain at each iteration.

`end.state` a character vector of length equal to the number of chains representing a description of the model state in the R dump format for each chain at the last iteration. This can be used as an initial values vector to restart a new simulation in the same place as the previous simulation ended.

`burnin` number of burnin iterations discarded before sampling.

`sample` number of sampled iterations.

`thin` the thinning interval in JAGS used for the chains.

`summary` the summary of each monitored variable from the combined chains, equivalent to summary(combine.mcmc(mcmc, collapse.chains=TRUE)).

`HPD` the 95% highest posterior density and median value of each monitored variable from the combined chains.

`psrf` the output of the Gelman Rubin dignostic (similar [but not exactly equivalent] to gelman.diag(mcmc)).

`autocorr` the output of the autocorrelation diagnostic (equivalent to autocorr.diag(mcmc)).

`trace` a list of lattice objects representing traceplots for each monitored variable, of class 'plotindpages'. Calling each individual element will result in the traceplot for that variable being shown, calling the entire list will result in all traceplots being shown in new windows (which may cause problems with your R session if there are several monitored variables). To override the individual window plotting behaviour (to combine plots and/or save the plots to a file), either change the class of each object using for(i in 1:nvar(results$mcmc)) class(results$trace[[i]]) <- 'trellis' and then combine using the c.trellis method in the latticeExtra package, or simply re-generate the plots using the raw mcmc output. Not produced if plots==FALSE.

`density` a list of lattice objects representing density plots for each monitored variable, of class 'plotindpages'. Calling each individual element will result in the density plot for that variable being shown, calling the entire list will result in all density plots being shown in new windows (which may cause problems with your R session if there are several monitored variables). To override the individual window plotting behaviour (to combine plots and/or save the plots to a file), either change the class of each object using for(i in 1:nvar(results$mcmc)) class(results$density[[i]]) <- 'trellis' and then combine using the c.trellis method in the latticeExtra package, or simply re-generate the plots using the raw mcmc output. Not produced if plots==FALSE.

`pd` the total effective number of parameters in the model calculated using the DIC module for JAGS. Returned only if monitor.pd=TRUE is supplied to the function (or if 'dic' is specified as a monitored variable).

`pd.i` the contribution of each parameter towards the total effective number of parameters calculated using the DIC module for JAGS. Returned only if monitor.pd.i=TRUE is supplied to the function.

`popt` the optimism of the expected deviance calculated using the DIC module for JAGS. Returned only if monitor.popt=TRUE is supplied to the function (or if 'dic' is specified as a monitored variable).

`dic` The Deviance Information Criterion (DIC) model fit statistics. Returned only if 'dic' is specified as a monitored variable. The default print method displays the

DIC calculated using both pd (dic) and popt (ped), using the deviance from the combined chains and for each individual chain. Separate values for the mean pd, sum popt and mean deviance is also listed. For more information on the types of DIC calculated, see the JAGS manual.

Author(s)

Matthew Denwood <matthew.denwood@glasgow.ac.uk>

See Also

autorun.jags,
run.jagsfile,
combine.mcmc,
testjags,
dump.format,
summary.mcmc,
gelman.diag,
autocorr.diag

Examples

```
# run a model to calculate the intercept and slope of the expression
# y = m x + c, assuming normal observation errors for y:

## Not run:

# Simulate the data
X <- 1:100
Y <- rnorm(length(X), 2*X + 10, 1)

# Model in the JAGS format
model <- "model {
for(i in 1 : N){
Y[i] ~ dnorm(true.y[i], precision);
true.y[i] <- (m * X[i]) + c;
}
m ~ dunif(-1000,1000);
c ~ dunif(-1000,1000);
precision ~ dexp(1);
}"

# Use dump.format to convert the data and initial values files
# into the R dump format, with explicit control over the random
# number generator used for each chain (optional):
data <- dump.format(list(X=X, Y=Y, N=length(X)))
inits1 <- dump.format(list(m=1, c=1, precision=1,
.RNG.name="base::Super-Duper", .RNG.seed=1))
```

```
inits2 <- dump.format(list(m=0.1, c=10, precision=1,
.RNG.name="base::Wichmann-Hill", .RNG.seed=2))

# Run the model and produce plots
results <- run.jags(model=model, monitor=c("m", "c", "precision"),
data=data, n.chains=2, inits=c(inits1,inits2), plots = TRUE)

# Density plots of the monitored variables:
results$density

# Analyse the results
results$summary


## End(Not run)
```

`run.jagsfile` *Read a User Specified Model in a WinBUGS Type Textfile or Character*

Description

Runs a user specified JAGS (similar to WinBUGS) model in a WinBUGS type textfile or character variable from within R, returning a list of the MCMC chain(s) along with optional convergence diagnostics, autocorrelation diagnostics and monitored variable summaries. JAGS is called using the lower level function `run.jags`, and `read.winbugs` is used to extract model and data definitions from the specified file or string.

Usage

```
run.jagsfile(path=stop("No path or model string supplied"),
datalist = NA, initlist = NA, n.chains=NA, data=NA,
model=NA, inits=NA, monitor=NA, call.jags=TRUE,
autorun=FALSE, ...)
```

Arguments

`path`	either a relative or absolute path to a textfile (including the file extension) containing a model in the JAGS language and possibly monitored variable names, data and/or initial values, or a character string of the same. May also be a vector of paths to different text files, possibly separately containing the model, data and intitial values. No default. The model must be started with the string 'model{' and ended with '}' on new lines. Data must be similarly started with 'data{', monitored variables with 'monitor{', and initial values as 'inits{', and all ended with '}'. If multiple models are found, all but the first one are ignored with a

warning. Multiple data blocks and monitor blocks are combined, multiple inits blocks are used for different chains. The model block may also contain automatically generated data and initial values variables using '#data# variable' and '#inits# variable', and more monitored variables using '#monitor# variable'. See `read.winbugs` for more information.

`datalist` an optional named list containing variables used as data, or alternatively a function (with no arguments) that returns a named list. If any variables are specified in the model block using '#data# variable', the value for the corresponding named variable is taken from datalist if present (or the result of datalist() if specified as a function which is useful for specifying randomly generated data), or the parent environment, or finally the global environment if not found anywhere else. Ignored if '#data# variable' is not used in the model block. Default NA.

`initlist` an optional named list containing variables used as initial values, or alternatively a function (with a single argument representing the chain number) that returns a named list. If any variables are specified in the model block using '#inits# variable', the value for the corresponding named variable is taken from initlist if present (or the result of datalist(chain.no) if specified as a function which allows both randomly generated initial values and different values for each chain), or the parent environment, or finally the global environment if not found anywhere else. Ignored if '#inits# variable' is not used in the model block. Note: different chains are all given the same starting values if specified as a named list or taken from any envirnoment; if different values are desired for each chain initlist should be specified as a function. Default NA.

`n.chains` the number of chains to use with the simulation. More chains will improve the sensitivity of the convergence diagnostic, but will cause the simulation to run more slowly. If NA, the number of chains will be taken from the number of inits blocks in the model file. If NA and no inits blocks are found, 2 chains are used with a warning. Default NA.

`data` OPTIONAL character vector in the R dump format (or named list) containing the data. If supplied (!=NA), all data in the model file is ignored. Default NA.

`model` OPTIONAL model in JAGS syntax. If supplied (!=NA), the model in the model file is ignored. Default NA.

`inits` OPTIONAL character vector(s) in the R dump format containing the initial value(s). If supplied (!=NA), all inits in the model file are ignored. Default NA.

`monitor` OPTIONAL character vector containing the monitored variables. If supplied (!=NA), all monitor statements in the model block are ignored. Default NA.

`call.jags` option results in either simulation being called if TRUE, or returns a named list of the data, model, initial values, monitored variables and number of chains (which can be supplied to run.jags) if FALSE.

`autorun` option to call autorun.jags rather than run.jags for the simulation, which allows automatic calculation of the necessary run length and convergence diagnostics. If TRUE, burnin, sample and check.conv are ignored. See also `autorun.jagsfile` for a wrapper for this function. Default FALSE.

`...` other options to be passed directly to run.jags (see `run.jags`) or autorun.jags (see `autorun.jags`).

Value

The output of run.jags (or autorun.jags if autorun==TRUE). See the help file run.jags or autorun.jags for more information.

Author(s)

Matthew Denwood <matthew.denwood@glasgow.ac.uk>

See Also

autorun.jagsfile,

run.jags,

autorun.jags,

read.winbugs

Examples

```
# run a model to calculate the intercept and slope of the expression
# y = m x + c, assuming normal observation errors for y:

## Not run:

# Model in the JAGS format

# Model in the JAGS format
model <- "model {
for(i in 1 : N){ #data# N
Y[i] ~ dnorm(true.y[i], precision); #data# Y
true.y[i] <- (m * X[i]) + c; #data# X
}
m ~ dunif(-1000,1000); #inits# m
c ~ dunif(-1000,1000);
precision ~ dexp(1);
#monitor# m, c, precision
}"

# Simulate the data
X <- 1:100
Y <- rnorm(length(X), 2*X + 10, 1)
N <- length(X)

initfunction <- function(chain) return(switch(chain,
"1"=list(m=-10), "2"=list(m=10)))

results <- run.jagsfile(model, n.chains=2, initlist=initfunction)

# Analyse the results
results$summary
## End(Not run)
```

runjags *Run Bayesian MCMC Models in the BUGS syntax from Within R*

Description

A set of functions to allow any user specified model to be run in JAGS from within R, returning the MCMC chains as R objects. Includes functions to read external WinBUGS type textfiles, and allows several ways of automatically specifying model data from existing R objects or R functions. Also includes functions to automatically calculate model run length, autocorrealtion and Gelman Rubin statistic diagnostics for all models to simplify the process of achieving chain convergence. Designed for maximum compatibility with WinBUGS syntax, although minor modification to existing .bug files will be required. Also provides functions for submission and retrieval of jobs (both JAGS runs and user-specified R code execution) to Apple Xgrid distributed computing clusters (Mac OS X machines only). Requires Just Another Gibbs Sampler (JAGS) for most functions, see: http://www-fis.iarc.fr/~martyn/software/jags/

Details

JAGS is a program which allows analysis of Bayesian models using Markov chain Monte Carlo (MCMC) simulation, and was developed by Martyn Plummer to be an alternative to BUGS that ran on UNIX systems as well as Windows systems (see: http://www-fis.iarc.fr/~martyn/software/jags/ for more information). The R package rjags is a native R interface to the JAGS library, and allows a far greater level of control and analysis of compiled models, which is often more useful for model development. This package was intended to provide additional functions to help automate the process of running models, including interpretation of WinBUGS type text files including data and initial values, automated retrieval of objects within the working environment to be passed as data or initial values to the model, automated convergence diagnostics, and automated collation and plotting of results. The package also includes functions for running JAGS models, and any other user specified R code, over Xgrid distributed computing clusters from within R (requires Mac OS X and access to an Xgrid system).

Author(s)

Matthew Denwood <matthew.denwood@glasgow.ac.uk>

See Also

run.jags for basic model runs

autorun.jags for automated running of models to convergence and automated calculation of necessary sample sizes

run.jagsfile and autorun.jagsfile for running of local WinBUGS style text files, as well as methods of automating retrieval of data and initial values from local text files or character strings

read.winbugs for translation of WinBUGS text files into JAGS compatible model, data and initial values files

combine.mcmc and dump.format for MCMC related tools

timestring, new_unique and ask for more general tools

xgrid.run.jags and xgrid.submit.jags for use of Xgrid clusters to run JAGS models remotely

xgrid.run and xgrid.submit for use of Xgrid clusters for remote execution of user specified R code

jags.model in the rjags package for fine control over the JAGS libraries

---

testjags *Analyse the System to Check That Jags Is Installed*

Description

Test the users system to determine the operating system, version of R installed, and version of JAGS installed. Some information is collected from other functions such as .platform and Sys.info. Used by the run.jags function.

Usage

```
testjags(jags=findjags(), silent=FALSE)
```

Arguments

jags	the system call or path for activating JAGS. Default calls findjags() to attempt to locate JAGS on your system automatically. In unix the system call should always be 'jags', in Windows a path to the JAGS executable or the enclosing /bin or /JAGS folder is required.
silent	should on-screen feedback be suppressed? Default FALSE.

Value

The following self-explanatory information is returned: os, JAGS.available (logical), JAGS.path, popen.support (internalisation of terminal output), JAGS.version, R.version, R.GUI, R.package.type, username.

Author(s)

Matthew Denwood <matthew.denwood@glasgow.ac.uk>

See Also

run.jags,

findjags

Examples

```
# Run the function to determine if JAGS is installed:
results <- testjags()
```

`timestring` *Calculate the Elapsed Time in Sensible Units*

Description

Function to calculate the elapsed time between 2 time periods (in seconds), or to calculate a number of seconds into a time measurement in more sensible units.

Usage

```
timestring(time1, time2=NA, units=NA, show.units=TRUE)
```

Arguments

`time1`	either the time index (from Sys.time()) at the start of the time period, or a length of time in seconds.
`time2`	either the time index (from Sys.time()) at the end of the time period, or missing data if converting a single length of time. Default NA.
`units`	either missing, in which case a sensible time unit is chosen automatically, or one of 's', 'm', 'h', 'd', 'w', 'y' to force a specific unit. Default NA.
`show.units`	if TRUE, then the time is returned with units, if FALSE then just an integer is returned. Default TRUE.

Value

A time measurement, with or without units.

Author(s)

Matthew Denwood <matthew.denwood@glasgow.ac.uk>

See Also

`Sys.time`

Examples

```
# time how long it takes to complete a task:

pre.time <- Sys.time()
for (i in 1:10000000) hold <- exp(100)  # PROCESS TO TIME
post.time <- Sys.time()
timestring(pre.time, post.time)

# Convert 4687 seconds into hours:
```

```
timestring(4687, units='hours', show.units=FALSE)
```

`xgrid.run` *Remote execution of user-specified R functions on Apple Xgrid*

Description

Allows arbitrary R code to be executed on Apple Xgrid distributed computing clusters and the results returned to the R session of the user. Jobs can either be run synchronously (the process will wait for the model to complete before returning the results) or asynchronously (the process will terminate on submission of the job and results are retrieved at a later time). Access to an Xgrid cluster with R (along with all packages required by the function) installed is required. Due to the dependance on Xgrid software to perform the underlying submission and retrieval of jobs, these functions can only be used on machines running Mac OS X. Further details of required environmental variables and the optional mgrid script to enable multi-task jobs can be found in the details section.

'xgrid.run' submits jobs to Xgrid that execute the function provided over the number of iterations specified, then intermittently retrieves the status of the job(s) and, if finished, retrieving and returning the results as an R list object.

'xgrid.submit' submits the job to xgrid, and returns the name of the started job (this is a convinience wrapper for xgrid.run with submitandstop=TRUE).

'xgrid.results' returns the results of a job started using 'xgrid.submit' in the current working directory. If the job is not complete the function will return the status of the job, or the results for completed threads (without deleting the job) if partial.retrieve=TRUE

'xapply' is a convinience wrapper for 'xgrid.run' which takes arguments akin to lapply

Usage

```
xgrid.run(f=function(iteration){}, niters,
   object.list=list(), file.list=character(0),
   threads=min(niters,100), arguments=as.list(1:niters),
   jobname=NA, wait.interval="10 min",
   xgrid.method=if(threads==1) 'simple' else
   if(!file.exists(Sys.which('mgrid'))) 'separatejobs'
   else 'separatetasks', Rpath='/usr/bin/R', Rbuild='64',
   cleanup = TRUE, submitandstop = FALSE, tempdir=!submitandstop,
   keep.files = FALSE, show.output = TRUE, max.filesize="1GB",
   sub.app=if(!file.exists(Sys.which('mgrid')))
   'xgrid -job submit -in "$indir"'
   else 'mgrid -t $ntasks -i "$indir"', sub.options="",
   sub.command=paste(sub.app, sub.options, '"$cmd"', sep=' '),
   ...)
```

```
xgrid.submit(f=function(iteration){}, niters,
   object.list=list(), file.list=character(0),
   threads=min(niters,100), arguments=as.list(1:niters),
   jobname=NA, xgrid.method=if(threads==1) 'simple' else
   if(!file.exists(Sys.which('mgrid'))) 'separatejobs'
   else 'separatetasks', Rpath='/usr/bin/R', Rbuild='64',
   show.output = TRUE, max.filesize="1GB",
   sub.app=if(!file.exists(Sys.which('mgrid')))
   'xgrid -job submit -in "$indir"'
   else 'mgrid -t $ntasks -i "$indir"', sub.options="",
   sub.command=paste(sub.app, sub.options, '"$cmd"', sep=' '),
   ...)

xgrid.results(jobname, partial.retrieve=FALSE,
   cleanup=!partial.retrieve, keep.files=FALSE, show.output=TRUE)

xapply(X, FUN, xgrid.options=list(), ...)
```

Arguments

f	the function to be iterated over on Xgrid. This must take at least 1 argument, the first of which represents the value of the 'arguments' list to be passed to the function for that iteration, which is the iteration number unless 'arguments' (or 'X' for xapply) is specified. Any other arguments to be passed to the function can be supplied as additional arguments to xgrid.run/xgrid.submit/xapply. The value(s) of interest should be returned by this function (an object of any class is permissable). No default.
niters	the total number of iterations over which to evaluate the function f. This can be less than the number of threads, in which case multiple iterations are evaluated serially as part of the same task. No default.
object.list	a named list of objects that will be copied to the global environment on Xgrid and so will be visible inside the function. Alternatively, this can be a character vector of objects, that will be looked for in the global environment, rather than a named list. All other objects in the current working directory will not be visible when the function is evaluated. THIS INCLUDES LIBRARIES WHICH MUST BE RE-CALLED WITHIN THE FUNCTION BEFORE USE. In order to use functions within an R library it is therefore necessary for the required library to be installed on the Xgrid nodes on which the job will be run. If not all nodes have the required libraries installed, you can use an ART script to ensure the job is sent only to machines that do (see the example provided below), or you can use mgrid to manually request certain nodes using the '-f -h <nodename>' options. Alternatively, text files containing R code can be included in the 'file.list' argument and source()d within the function. Default blank list (no objects copied).
file.list	a vector of filenames representing files in the current working directory that will be copied to the working directory of the executed function. This allows R code

to be source()d, datasets to be loaded, and compiled code to be dynamically linked within the function, among other things. Default none.

`threads` the number of threads to generate for the job. Threads is taken to mean jobs if xgrid.method is 'separatejobs' or tasks if xgrid.method is 'separatetasks'. Each thread is sent to a separate node for execution, so the more threads there are the faster the job will finish (unless the number of threads exceeds the number of available nodes). A very large number of threads may cause problems with the Xgrid controller, hence the ability to set fewer threads than iterations. Functions that return objects of a very large size should use a large number of threads and use the xgrid.method 'separatejobs' to minimise the total size of objects returned by each xgrid job. Default is equal to the number of iterations if this is less than 100, or 100 otherwise.

`arguments` a list of values to be passed as the first argument to the function, with each element of the list specifying the value at that iteration. Default is as.list(1:niters) which passes only the iteration number to the function.

`jobname` for all functions except xgrid.results.jags, the jobname can be provided to make identification of the job using Xgrid Admin easier. If none is provided, then one is generated using a combination of the username and hostname of the submitting machine. If the provided jobname is already used by a file/folder in the working directory, then the name is altered to be unique using new_unique(). For xgrid.results.jags, the jobname must be supplied to match the jobname value returned by xgrid.submit.jags(file) during job submission.

`wait.interval` when running xgrid jobs synchronously, the waiting time between retrieving the status of the job. If the job is found to be finished on retrieving the status then results are returned, otherwise the function waits for 'wait.interval' before repeating the process. Time units of seconds, minutes, hours, days or weeks can be specified. If no units are given the number is assumed to represent minutes. Default "10 min".

`xgrid.method` the method of submitting the work to Xgrid - one of 'simple', 'separatejobs' or 'separatetasks'. The former runs all chains on a single node, whereas 'separatejobs' runs all chains as individual xgrid jobs and 'separatetasks' runs all chains as individual tasks within the same job (this makes the job information in Xgrid Admin easier to read). The latter method requires a submission script that is capable of supporting multi-task jobs, such as the mgrid script included with the runjags package (see the details section for more details and installation instructions). If each chain is likely to return a large amount of information then 'separatejobs' should be used; this is because jobs are retrieved individually which reduces the chances of overloading the Xgrid controller. Default 'simple' if threads==1; otherwise 'separatetasks' if mgrid is available or 'separatejobs' if not.

`Rpath` the path to the R executable on the xgrid machines. If not all machines on the xgrid cluster have R (or a required package) installed then it is possible to use an ART script to ensure the job is sent to only machines that do - see the examples section for details. Default '/usr/bin/R' (this is the default install location for R).

`Rbuild` the preferred binary of R to invoke. '64' results in 'Rpath64' (if it exists), '32' in 'Rpath32' (if it exists) and '' (or either of '32' or '64' if they are not found)

results in Rpath. Notice that this indicates a preference, not a certainty - if the indicated build is not avalable then another will be used. Also note that specifying '64' may be ignored for PPC nodes depending on what version of R they are running (you can ensure only intel nodes are used with mgrid using sub.options='-c intel'). Default ".

`partial.retrieve`
: for xgrid.results, option to retrieve results of partially completed jobs. By default makes cleanup FALSE. Default TRUE.

`cleanup`
: option to delete the job(s) from Xgrid after retrieving result.

`submitandstop`
: controls whether job should be run synchronously (submitandstop=FALSE), in which case the process will wait for the model to complete before returning the results, or asynchronously (submitandstop=TRUE), in which case the process will terminate on submission of the job and results are retrieved at a later time. Default for xgrid.run is FALSE. xgrid.submit is a wrapper to xgrid.run with submitandstop=TRUE.

`tempdir`
: for xgrid.run, option to use the temporary directory as specified by the system rather than creating files in the working directory. Any files created in the temporary directory are removed when the function exits. A temporary directory cannot be used for xgrid.submit. Default TRUE when running the job synchronously.

`keep.files`
: option to keep the folder with files needed to run the job rather than deleting it, or copy the folder to the working directory before exiting if tempdir=TRUE. This may be useful for attempting to bug fix failing jobs. Default FALSE.

`show.output`
: option to print the output of the function (obtained using cat, writeLine or print for example) at each iteration after retrieving the job(s) from xgrid. If FALSE, the output is suppressed. Default TRUE.

`max.filesize`
: the maximum total size of the objects produced by the function for each thread if xgrid.method=separatejobs, or for the entire job if xgrid.method=separatetasks. This is a failsafe designed to prevent attempted transfer of huge files bringing the xgrid controller down. If the maximum size is exceeded for a thread or job then the results are erased for all iterations within that thread or job, and the job will likely have to be re-submitted. If each chain is likely to return a large amount of information, then 'separatejobs' should be used because jobs are retrieved individually which reduces the chances of overloading the Xgrid controller. The object.list is also checked to ensure it complies with the maximum size, but the file.list and any objects saved to the working directory by the function are NOT automatically cheked. Units can be provided as either "MB" or "GB". Default "1GB".

`sub.app`
: the submission application or script to use for job running/submission. The inbuilt Xgrid application supports most options, but greater functionality is provided by the mgrid script (see the details section for more information and installation instructions). Any other custom script can be used with the requirements that it submit the job provided and print the Xgrid job ID to screen before exiting (as the only numerical value printed), or alternatively the script may submit the job and create a 'jobid.txt' file in the working directory containing the job id. If

xgrid.method is 'separatejobs' then the argument may be of length equal to the number of chains, in which case each job is submitted using a different application/script. Paths with spaces in them must be quoted when the command is passed to the shell (this may mean escaping quotes if necessary). Default uses mgrid if installed, otherwise 'xgrid -job submit'.

`sub.options` one or more option flags to be passed through to the submission application (as a character string). Examples include ART scripts, email on job completion, and when using the mgrid script many other possibilities (see the details section). When providing links to files as part of the command, all links must be absolute (ie start with / or ~) as xgrid/mgrid will be will not be called in the working directory, and paths with spaces must be quoted. If xgrid.method is 'separatejobs' then the argument may be of length equal to the number of chains, in which case each job receives a different set of options. Some options require the Xgrid controller to be running OS X Leopard (10.5) or later. Default none.

`sub.command` the actual command to be executed using system() to submit the job. Changing this results in sub.app and sub.options being ignored, and is probably the best option to use for custom submission scripts (see the sub.app argument for the requirements for custom scripts). The environmental variables $cmd (the name of the BASH script to be run), $ntasks (the number of tasks), $job (the job number for multiple jobs), and $indir (the input directory) will be available to the script. For multiple tasks, the custom script should ensure that the task number is supplied as the (only) argument to the BASH script (requires xgrid.method="separatetasks" to function). If xgrid.method is 'separatejobs' then the argument may be of length equal to the number of chains, in which case each job receives a different command. Paths with spaces in them must be quoted when the command is passed to the shell (this may mean escaping quotes if necessary). Default uses the values of sub.app and sub.options.

`X` for xapply, a vector (atomic or list) over which to apply the function provided. Equivalent to 'arguments' for xgrid.run, with niters = length(X).

`FUN` for xapply, the function to be passed to xgrid.run as 'f'.

`xgrid.options`
for xapply, any arguments (with the exception of 'f', 'niters' and 'arguments' which are ignored) to be passed to xgrid.run.

`...` additional arguments to be passed to the function provided.

Details

These functions allow JAGS models to be run on Xgrid distributed computing clusters from within R using the same syntax as required to run the models locally. All the functionality could be replicated by saving all necessary objects to files and using the Xgrid command line utility to submit and retrieve the job manually; these functions merely provide the convenience of not having to do this manually. Xgrid support is only available on Mac OS X machines.

The xgrid controller hostname and password must be set as environmental variables. The command line version of R knows about environmental variables set in the .profile file, but unfortunately the GUI version does not and requires them to be set from within R using:

Sys.setenv(XGRID_CONTROLLER_HOSTNAME="<hostname>")

Sys.setenv(XGRID_CONTROLLER_PASSWORD="<password>")

(These lines could be copied into your .Rprofile file for a 'set and forget' solution)

All functions can be run using the built-in xgrid commands, however some added functionality (including multi-tasks jobs to enable the 'separatetasks' method) is provided by the 'mgrid.sh' BASH shell script which is included with the runjags package (in the 'inst/xgrid' folder for the package source or the 'xgrid' folder for the installed package). More details about this script is given at the top of the mgrid.sh file. To install (optional), see the `install.mgrid` function.

Value

For xgrid.submit, a list containing the jobname (which will be required by xgrid.results to retrieve the job) and the job ID(s) for use with the xgrid command line facilities. For xgrid.run and xgrid.results, the output of the function over all iterations is returned as a list, with each element of the list representing the results at each iteration. If the function returned an error, then the error will be held in the list as the return value at the iteration that returned the error. If the function returns an object that exceeds the 'max.filesize' when combined with the results for other iterations in that job (or greater than max.filesize/threads for multi-task jobs), the results for that thread are replaced with an error message (this is to prevent the xgrid controller crashing due to transferring large files). The xapply function returns as xgrid.run (or xgrid.submit if xgrid.options=list(submitandstop=TRUE) in which case the results can be retrieved using xgrid.results).

Author(s)

Matthew Denwood <matthew.denwood@glasgow.ac.uk>

See Also

`xgrid.run.jags` for functions to run JAGS models on Xgrid, or `run.jags` to do so locally.

`install.mgrid` to install the mgrid script.

mclapply and mcparallel in the multicore package for parallel execution of code over multiple local cores.

Examples

```
# A basic example of synchronous running of code over 100 iterations,
# split up between 10 tasks (or 10 jobs if mgrid is not installed):

## Not run:

# The function to evaluate:
f <- function(iteration){
# All objects supplied to object.list will be visible here, but
# remember to call all necessary libraries within the function

cat("Running iteration", iteration, "\n")
# Some lengthy code evaluation....

output <- rpois(10, iteration)
```

```
return(output)
}

# Run the function on xgrid for 100 iterations split between 10 machines:
results <- xgrid.run(f, niters=100, threads=10)


## End(Not run)



# A basic example of xapply to calculate the mean of a list of numbers:

## Not run:

# A list of 3 datasets from which to calculate the mean:
datasets <- list(c(1,5,6,NA), c(9,2,NA,0), c(-1,4,10,20))

# Standard lapply syntax:
results1 <- lapply(datasets, mean, na.rm=TRUE)

# Equivalent xapply syntax:
results2 <- xapply(datasets, mean,
xgrid.options=list(wait.interval='15s'), na.rm=TRUE)

# Or submit the job:
id <- xapply(datasets, mean, xgrid.options=list(submitandstop=TRUE),
na.rm=TRUE)
# And retrieve the results:
results3 <- xgrid.results(id)


## End(Not run)



# Any packages required by the function need to be installed on the
# nodes the function is run on.  This function retrieves information
# about the available packages on each of the node names provided:

## Not run:

# The name of one or more nodes to get information about:
nodenames <- c("mynode", "guestnode", "othernode")

# Run the job:
results <- xgrid.run(function(i){
return(installed.packages()[,'Version'])
},
niters=length(nodenames), threads=length(nodenames),
wait.interval="10 seconds", xgrid.method='separatejobs',
sub.options=paste("-f -h '", nodenames, "'", sep=""),
show.output=FALSE)
```

```
# Make the names match up to the statistics:
names(results) <- nodenames

# Show the available packages and their versions for each node:
results


## End(Not run)


# An example of running an Xgrid job within another Xgrid job, using
# xgrid.submit to submit a job that runs a JAGS model to convergence
# using xgrid.autorun.jags:

## Not run:

# Create an ART script to make sure that (a) R is installed,
# (b) JAGS is installed, and (c) the runjags package is installed
# on the node:
cat('#!/bin/bash
#!/bin/bash
if [ ! -f /usr/bin/R ]; then
echo 0
exit 0
fi
if [ ! -f /usr/local/bin/jags ]; then
echo 0
exit 0
fi
/usr/bin/R --slave -e "suppressMessages(r<-require(runjags,quietly=TRUE));cat(r*1,fill=TRUE)
exit 0
', file='runjagsART.sh')

# Some data etc we will need for the model:
library(runjags)

X <- 1:100
Y <- rnorm(length(X), 2*X + 10, 1)
data <- dump.format(list(X=X, Y=Y, N=length(X)))

# Model in the JAGS format
model <- "model {
for(i in 1 : N){
Y[i] ~ dnorm(true.y[i], precision);
true.y[i] <- (m * X[i]) + c;
}
m ~ dunif(-1000,1000);
c ~ dunif(-1000,1000);
precision ~ dexp(1);
}"

# Get the Xgrid controller hostname and password to be passed
# to the slave job:
```

```
hostname <- Sys.getenv('XGRID_CONTROLLER_HOSTNAME')
password <- Sys.getenv('XGRID_CONTROLLER_PASSWORD')

# The function we are going to call on xgrid:
f <- function(iteration){
# Make sure the necessary environmental variables are set:
Sys.setenv(XGRID_CONTROLLER_HOSTNAME=hostname)
Sys.setenv(XGRID_CONTROLLER_PASSWORD=password)

# Call the library on the node:
library(runjags)

# Use xgrid.autorun.jags to run 2 chains until convergence:
results <- xgrid.autorun.jags(model=model,
monitor=c("m", "c", "precision"), data=data, n.chains=2,
inits=list(list(.RNG.name='base::Wichmann-Hill'),
list(.RNG.name='base::Marsaglia-Multicarry')),
plots = FALSE, xgrid.method='separatejobs',
wait.interval='1 min', jobname='xgridslavejob')

return(results)
}

# Submit the function to xgrid using our ART script to ensure the
# node can handle the job (the ART script path must be specified as
# an absolute link as xgrid won't be called in the current working
# directory, and all paths must be enclosed in quotes to preserve
# spaces):
name <- xgrid.submit(f, object.list=list(X=X, Y=Y, model=model,
data=data, hostname=hostname, password=password), threads=1,
niters=1, sub.options=if(!file.exists(Sys.which('mgrid')))
paste('-art "', getwd(), '/runjagsART.sh"', sep='') else
paste('-a "', getwd(), '/runjagsART.sh"', sep=''),
xgrid.method='simple')
# Cleanup (remove runjagsART file):
unlink('runjagsART.sh')

# Get the results once it is finished:
results <- xgrid.results(name)$iteration.1


## End(Not run)



## Not run:

# Subit an xgrid job just to see which packages are installed
# on a particular machine.

# Ensure mgrid is installed:
if(!file.exists(Sys.which('mgrid'))) install.mgrid()
```

```
# A function to harvest details of R version and installed packages:
f <- function(i){

archavail <- any(dimnames(installed.packages())[[2]]=='Archs')

# To deal with older versions of R:
if(archavail){
packagesinst <- installed.packages()[,c('Version', 'Archs', 'Built')]
}else{
packagesinst <- installed.packages()[,c('Version', 'OS_type', 'Built')]
}

Rinst <- unlist(R.version[c('version.string', 'arch', 'platform')])
names(Rinst) <- c('Version', 'Archs', 'Built')
return(rbind(R=Rinst, packagesinst))

}

# Or to get more details about a particular package:
g <- function(i){
p <- library(help='bayescount')
return(p$info)
}

# Get the information back from 2 specific machines called 'newnode1'
# and 'newnode2':

results <- xgrid.run(f, niters=2, threads=2,
sub.options='-h newnode1:newnode2', wait.interval='15 seconds')


## End(Not run)
```

`xgrid.run.jags` *Run a JAGS Model using an Xgrid distributed computing cluster from*

Description

Extends the functionality of the (auto)run.jags(file) family of functions to use with Apple Xgrid distributed computing clusters. Jobs can either be run synchronously using xgrid.(auto)run.jags(file) in which case the process will wait for the model to complete before returning the results, or asynchronously using xgrid.submit.jags(file) in which case the process will terminate on submission of the job and results are retrieved at a later time using xgrid.results.jags. The latter function can also be used to check the progress of incomplete simulations without stopping or retrieving the full job. Access to an Xgrid cluster with JAGS (although not necessarily R) installed is required. Due to the dependance on Xgrid software to perform the underlying submission and retrieval of jobs, these

functions can only be used on machines running Mac OS X. Further details of required environmental variables and the optional mgrid script to enable multi-task jobs can be found in the details section.

Usage

```
xgrid.run.jags(wait.interval="10 min", xgrid.method='simple',
jagspath='/usr/local/bin/jags', jobname=NA, cleanup=TRUE,
sub.app=if(!file.exists(Sys.which('mgrid')))
'xgrid -job submit -in "$indir"'
else 'mgrid -t $ntasks -i "$indir"', sub.options="",
sub.command=paste(sub.app, sub.options, '"$cmd"',
sep=' '), ...)

xgrid.run.jagsfile(wait.interval="10 min", xgrid.method='simple',
jagspath='/usr/local/bin/jags', jobname=NA, cleanup=TRUE,
sub.app=if(!file.exists(Sys.which('mgrid')))
'xgrid -job submit -in "$indir"'
else 'mgrid -t $ntasks -i "$indir"', sub.options="",
sub.command=paste(sub.app, sub.options, '"$cmd"',
sep=' '), ...)

xgrid.autorun.jags(wait.interval="10 min", xgrid.method='simple',
jagspath='/usr/local/bin/jags', jobname=NA, cleanup=TRUE,
sub.app=if(!file.exists(Sys.which('mgrid')))
'xgrid -job submit -in "$indir"'
else 'mgrid -t $ntasks -i "$indir"', sub.options="",
sub.command=paste(sub.app, sub.options, '"$cmd"',
sep=' '), ...)

xgrid.autorun.jagsfile(wait.interval="10 min", xgrid.method='simple',
jagspath='/usr/local/bin/jags', jobname=NA, cleanup=TRUE,
sub.app=if(!file.exists(Sys.which('mgrid')))
'xgrid -job submit -in "$indir"'
else 'mgrid -t $ntasks -i "$indir"', sub.options="",
sub.command=paste(sub.app, sub.options, '"$cmd"',
sep=' '), ...)

xgrid.submit.jags(xgrid.method='simple', jagspath='/usr/local/bin/jags',
jobname=NA, sub.app=if(!file.exists(Sys.which('mgrid')))
'xgrid -job submit -in "$indir"' else 'mgrid -t $ntasks -i "$indir"',
sub.options="", sub.command=paste(sub.app, sub.options, '"$cmd"',
sep=' '), ...)


xgrid.submit.jagsfile(xgrid.method='simple',
jagspath='/usr/local/bin/jags',
jobname=NA, sub.app=if(!file.exists(Sys.which('mgrid')))
```

```
'xgrid -job submit -in "$indir"' else 'mgrid -t $ntasks -i "$indir"',
sub.options="", sub.command=paste(sub.app, sub.options, '"$cmd"',
sep=' '), ...)

xgrid.results.jags(jobname, cleanup=TRUE, ...)
```

Arguments

`wait.interval`
: when running xgrid jobs synchronously, the waiting time between retrieving the status of the job. If the job is found to be finished on retrieving the status then results are returned, otherwise the function waits for 'wait.interval' before repeating the process. Time units of seconds, minutes, hours, days or weeks can be specified. If no units are given the number is assumed to represent minutes. Default "10 min".

`xgrid.method`
: the method of submitting the simulation to Xgrid - one of 'simple', 'separatejobs' or 'separatetasks'. The former runs all chains on a single node, whereas 'separatejobs' runs all chains as individual xgrid jobs and 'separatetasks' runs all chains as individual tasks within the same job (this makes the job information in Xgrid Admin easier to read). Note that the 'seperatejobs' and 'seperatetasks' methods use separate JAGS instances to speed up execution of models with multiple chains, but cannot be used with monitor.pd, monitor.pd.i or monitor.popt. Each chain is specified using a different random number generator (.RNG.name) for up to 4 chains (the number of different RNG available in JAGS), unless .RNG.name is specified in the initial values. Because each chain uses a separate JAGS instance, JAGS has no way of ensuring independence between multiple chains using the same random number generator (as would normally be done when calling a single JAGS instance with multiple chains). Using more than 4 chains with the 'separatejobs' or 'separatetasks' method without the use of new RNG factories may therefore produce dependence between chains, and is not recommended (a warning is given if trying to do so). Also, the 'separatetasks' method requires a submission script that is capable of supporting multi-task jobs, such as the mgrid script included with the runjags package (see the details section for more details and installation instructions). If each chain is likely to return a large amount of information then 'separatejobs' should be used in preference to 'separatetasks'; this is because jobs are retrieved individually which reduces the chances of overloading the Xgrid controller. Default 'simple'.

`method`
: the method with which to call JAGS; one of 'simple', 'interruptible' or 'parallel'. The former runs JAGS as a foreground process (the default behaviour for runjags < 0.9.6), 'interruptible' allows the JAGS process to be terminated immediately using the interrupt signal 'control-c' (terminal/console versions of R only), and 'parallel' runs each chain as a separate process on a separate core. Note that the latter uses separate JAGS instances to speed up execution of models with multiple chains (at the expense of using more RAM), but cannot be used with monitor.pd, monitor.pd.i or monitor.popt. Each chain is specified using a different random number generator (.RNG.name) for up to 4 chains (the number of different RNG available in JAGS), unless .RNG.name is specified

in the initial values. Using more than 4 chains without the use of new RNG factories may produce dependence between chains, and is not recommended (a warning is given if trying to do so). Only the 'simple' method is available for Windows. On machines running Mac OS X and with access to an Apple Xgrid cluster, the method may be a list with an element 'xgrid.method="simple"' (see xgrid.run.jags for more information). Default 'interruptible' on terminal/console versions of R, or 'simple' on GUI versions of R or when running over xgrid (methods other than 'simple' require the use of 'ps' which is not available when running jobs as 'nobody' via xgrid).

`jagspath` the path to the JAGS executable on the xgrid machines. Note that /usr/local/bin is not included in the path when running Xgrid jobs, so it is safer to provide the full path. If not all machines on the xgrid cluster have JAGS installed then it is possible to use an ART script to ensure the job is sent to only machines that do - see the examples section for details. Default '/usr/local/bin/jags' (this is the default install location for JAGS).

`jobname` for all functions except xgrid.results.jags, the jobname can be provided to make identification of the job using Xgrid Admin easier. If none is provided, then one is generated using a combination of the username and hostname of the submitting machine. If the provided jobname is already used by a file/folder in the working directory, then the name is altered to be unique using new_unique(). For xgrid.results.jags, the jobname must be supplied to match the jobname value returned by xgrid.submit.jags(file) during job submission.

`cleanup` option to delete the job(s) from Xgrid after retrieving result. Default TRUE.

`sub.app` the submission application or script to use for job running/submission. The inbuilt Xgrid application supports most options, but greater functionality is provided by the mgrid script (see the details section for more information and installation instructions). Any other custom script can be used with the requirements that it submit the job provided and print the Xgrid job ID to screen before exiting (as the only numerical value printed), or alternatively the script may submit the job and create a 'jobid.txt' file in the working directory containing the job id. If xgrid.method is 'separatejobs' then the argument may be of length equal to the number of chains, in which case each job is submitted using a different application/script. Paths with spaces in them must be quoted when the command is passed to the shell (this may mean escaping quotes if necessary). Default uses mgrid if installed, otherwise 'xgrid -job submit'.

`sub.options` one or more option flags to be passed through to the submission application (as a character string). Examples include ART scripts, email on job completion, and when using the mgrid script many other possibilities (see the details section). When providing links to files as part of the command, all links must be absolute (ie start with / or ~) as xgrid/mgrid will be will not be called in the working directory, and paths with spaces must be quoted. If xgrid.method is 'separatejobs' then the argument may be of length equal to the number of chains, in which case each job receives a different set of options. Some options require the Xgrid controller to be running OS X Leopard (10.5) or later. Default none.

`sub.command` the actual command to be executed using system() to submit the job. Changing this results in sub.app and sub.options being ignored, and is probably the best option to use for custom submission scripts (see the sub.app argument for

the requirements for custom scripts). The environmental variables $cmd (the name of the BASH script to be run), $ntasks (the number of tasks), $job (the job number for multiple jobs), and $indir (the input directory) will be available to the script. For multiple tasks, the custom script should ensure that the task number is supplied as the (only) argument to the BASH script (requires xgrid.method="separatetasks" to function). If xgrid.method is 'separatejobs' then the argument may be of length equal to the number of chains, in which case each job receives a different command. Paths with spaces in them must be quoted when the command is passed to the shell (this may mean escaping quotes if necessary). Default uses the values of sub.app and sub.options.

`...` other options to be passed to the (auto)run.jags(file) functions as if the model were being run locally. The following options to be applied after running the simulation can be specified to xgrid.results.jags, and will be ignored for other functions: keep.jags.files, check.conv, plots, psrf.target, normalise.mcmc, check.stochastic, silent.jags

Details

These functions allow JAGS models to be run on Xgrid distributed computing clusters from within R using the same syntax as required to run the models locally. All the functionality could be replicated by saving all necessary objects to files and using the Xgrid command line utility to submit and retrieve the job manually; these functions merely provide the convenience of not having to do this manually. Xgrid support is only available on Mac OS X machines.

The xgrid controller hostname and password must be set as environmental variables. The command line version of R knows about environmental variables set in the .profile file, but unfortunately the GUI version does not and requires them to be set from within R using:

Sys.setenv(XGRID_CONTROLLER_HOSTNAME="<hostname>")

Sys.setenv(XGRID_CONTROLLER_PASSWORD="<password>")

(These lines could be copied into your .Rprofile file for a 'set and forget' solution)

All functions can be run using the built-in xgrid commands, however some added functionality (including multi-tasks jobs to enable the 'separatetasks' method) is provided by the 'mgrid.sh' BASH shell script which is included with the runjags package (in the 'inst/xgrid' folder for the package source or the 'xgrid' folder for the installed package). More details about this script is given at the top of the mgrid.sh file. To install (optional), see the `install.mgrid` function.

Value

For xgrid.submit.jags and xgrid.submit.jagsfile, a list containing the jobname (which will be required by xgrid.results.jags to retrieve the job) and the job ID(s) for use with the xgrid command line facilities. For all other functions, the results of the simulation are returned as with the respective (auto)run.jags(file) functions.

Author(s)

Matthew Denwood <matthew.denwood@glasgow.ac.uk>

See Also

run.jags, autorun.jags and run.jagsfile for more information on JAGS models.

xgrid.run for functions to execute user-specified functions on Xgrid.

install.mgrid to install the mgrid script.

Examples

```
# run a simple model on Xgrid using a single job:

## Not run:

# Ensure the required environmental variables are set:
Sys.setenv(XGRID_CONTROLLER_HOSTNAME="<hostname>")
Sys.setenv(XGRID_CONTROLLER_PASSWORD="<password>")

# Simulate the data
X <- 1:100
Y <- rnorm(length(X), 2*X + 10, 1)

# Model in the JAGS format
model <- "model {
for(i in 1 : N){
Y[i] ~ dnorm(true.y[i], precision);
true.y[i] <- (m * X[i]) + c;
}
m ~ dunif(-1000,1000);
c ~ dunif(-1000,1000);
precision ~ dexp(1);
}"

# Run the model synchronously using the 'simple' method
# and a wait interval of 1 minute:
results <- xgrid.run.jags(xgrid.method='simple',
wait.interval='1 min', model=model, monitor=c("m", "c",
"precision"), data=list(N=length(X), X=X, Y=Y), n.chains=2,
plots = FALSE)

# Analyse the results:
results$summary


## End(Not run)


# Submit a job to xgrid and (later) retrieve the results.  Use an
# ART script to ensure the job is only sent to nodes with JAGS installed:

## Not run:

# Ensure the required environmental variables are set:
```

```
Sys.setenv(XGRID_CONTROLLER_HOSTNAME="<hostname>")
Sys.setenv(XGRID_CONTROLLER_PASSWORD="<password>")

# Create the ART script we need to ensure JAGS is installed:
cat('#!/bin/bash
if [ -f /usr/local/bin/jags ]; then
echo 1
else
echo 0
fi
', file='jagsART.sh')

# Simulate the data
X <- 1:100
Y <- rnorm(length(X), 2*X + 10, 1)

# Model in the JAGS format
model <- "model {
for(i in 1 : N){
Y[i] ~ dnorm(true.y[i], precision);
true.y[i] <- (m * X[i]) + c;
}
m ~ dunif(-1000,1000);
c ~ dunif(-1000,1000);
precision ~ dexp(1);
}"

# Run the model asynchronously (the ART script path must
# be specified as an absolute link as xgrid won't be called
# in the current working directory, and all paths must be
# enclosed in quotes to preserve spaces):
name <- xgrid.submit.jags(xgrid.method='separatejobs',
sub.options=if(!file.exists(Sys.which('mgrid'))) paste('-art
"', getwd(), '/jagsART.sh"', sep='') else paste('-a "', getwd(),
'/jagsART.sh"', sep=''), model=model, monitor=c("m", "c", "precision"),
data=list(N=length(X), X=X, Y=Y), n.chains=2, plots = FALSE,
inits=list(list(.RNG.name='base::Wichmann-Hill'),
list(.RNG.name='base::Marsaglia-Multicarry')))

# Cleanup (remove jagsART file):
unlink('jagsART.sh')

# Retrieve the results:
results <- xgrid.results.jags(name)

## End(Not run)



# Autorun a model to convergence using separate tasks on xgrid.
# Ensure the tasks are sent to the 2 fastest nodes (called 'Bugati'
# and 'McLaren') in our (fictional) cluster using arguments to mgrid.
```

```
## Not run:

# Ensure the required environmental variables are set:
Sys.setenv(XGRID_CONTROLLER_HOSTNAME="<hostname>")
Sys.setenv(XGRID_CONTROLLER_PASSWORD="<password>")

# Ensure mgrid is installed:
if(!file.exists(Sys.which('mgrid'))) install.mgrid()

# Simulate the data
X <- 1:100
Y <- rnorm(length(X), 2*X + 10, 1)

# Model in the JAGS format
model <- "model {
for(i in 1 : N){
Y[i] ~ dnorm(true.y[i], precision);
true.y[i] <- (m * X[i]) + c;
}
m ~ dunif(-1000,1000);
c ~ dunif(-1000,1000);
precision ~ dexp(1);
}"

# Run the model synchronously using the 'separatetasks' method and
# a wait interval of 1 minute:
results <- xgrid.autorun.jags(xgrid.method='separatetasks',
wait.interval='1 min', sub.options='-h "Bugati:McLaren"',
model=model, monitor=c("m", "c", "precision"),
data=list(N=length(X), X=X, Y=Y), n.chains=2,
inits=list(list(.RNG.name='base::Wichmann-Hill'),
list(.RNG.name='base::Marsaglia-Multicarry')), plots = FALSE)


## End(Not run)
```

Index

∗Topic **methods**
ask, 2
combine.mcmc, 12
dump.format, 14
findjags, 15
install.mgrid, 16
new_unique, 20
read.winbugs, 21
runjags, 33
testjags, 34
timestring, 35
xgrid.run, 36
xgrid.run.jags, 45
∗Topic **models**
autorun.jags, 3
autorun.jagsfile, 10
run.jags, 24
run.jagsfile, 30

ask, 2, *21*, *34*
autocorr.diag, *9*, *29*
autorun.JAGS *(autorun.jags)*, 3
autorun.jags, 3, *11*, *29*, *31–33*, *50*
autorun.JAGSfile *(autorun.jagsfile)*, 10
autorun.jagsfile, *9*, 10, *31–33*

combine.MCMC *(combine.mcmc)*, 12
combine.mcmc, *9*, 12, *29*, *33*

dump.format, 14, *29*, *33*

findJAGS *(findjags)*, 15
findjags, 15, *34*

gelman.diag, *9*, *29*

install.mgrid, 16, *41*, *49, 50*

jags.model, *34*

menu, *3*

new_unique, 20, *34*

raftery.diag, *5*, *9*
read.WinBUGS *(read.winbugs)*, 21
read.winbugs, *10, 11*, 21, *30–33*
readline, *3*
run.JAGS *(run.jags)*, 24
run.jags, *3*, *9*, *11*, *13*, *15, 16*, 24, *30–34*, *41*, *50*
run.JAGSfile *(run.jagsfile)*, 30
run.jagsfile, *10, 11*, *21, 22*, *29*, 30, *33*, *50*
runJAGS *(runjags)*, 33
runjags, 33
runJAGS-package *(runjags)*, 33
runjags-package *(runjags)*, 33

summary.mcmc, *29*
Sys.time, *35*

testJAGS *(testjags)*, 34
testjags, *13*, *16*, *29*, 34
timestring, *34*, 35

xapply *(xgrid.run)*, 36
xgrid.autorun.JAGS *(xgrid.run.jags)*, 45
xgrid.autorun.jags *(xgrid.run.jags)*, 45
xgrid.autorun.JAGSfile *(xgrid.run.jags)*, 45
xgrid.autorun.jagsfile *(xgrid.run.jags)*, 45
xgrid.results *(xgrid.run)*, 36
xgrid.results.JAGS *(xgrid.run.jags)*, 45
xgrid.results.jags *(xgrid.run.jags)*, 45
xgrid.run, *20*, *34*, 36, *50*

xgrid.run.JAGS *(xgrid.run.jags)*, 45
xgrid.run.jags, *7*, *20*, *27*, *34*, *41*, 45, *48*
xgrid.run.JAGSfile *(xgrid.run.jags)*, 45
xgrid.run.jagsfile *(xgrid.run.jags)*, 45
xgrid.submit, *34*
xgrid.submit *(xgrid.run)*, 36
xgrid.submit.JAGS *(xgrid.run.jags)*, 45
xgrid.submit.jags, *34*
xgrid.submit.jags *(xgrid.run.jags)*, 45
xgrid.submit.JAGSfile *(xgrid.run.jags)*, 45
xgrid.submit.jagsfile *(xgrid.run.jags)*, 45